Controlling Multibody Dynamics via Browsing and Time Reversal

Christopher D. Twigg

CMU-CS-08-130
May 2008

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Doug L. James, Chair
Jessica K. Hodgins
James J. Kuffner
Zoran Popović, University of Washington

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

This research was sponsored by a NVIDIA Graduate Fellowship, Carnegie Mellon University, National Science Foundation Grant CAREER-0430528, Pixar, the Alfred P. Sloan Foundation, hardware donations by NVIDIA and Intel, and Maya licenses donated by Autodesk

Keywords: rigid body dynamics, frictional contact, linear complementarity, ill-posedness, inverse problems, motion planning, interactive animation, control, browsing, data-driven animation

For my family.

Acknowledgments

I would first like to thank my parents, who have offered support and encouragement and dared me to chase my dreams.

I would next like to thank my advisor, Doug James, who has consistently pushed me toward ever higher levels of mathematical rigor and encouraged me to take on difficult problems.

I would like to thank my committee, Jessica Hodgins, James Kuffner, and Zoran Popović, for their valuable feedback, questions, and insights.

The members of the Carnegie Mellon graphics lab have suffered alongside me through countless SIGGRAPH deadlines and have always been ready to chat about anything from life to linear algebra. Thanks to Jernej Barbič, Adam Bargteil, Kiran Bhat, Chris Cameron, Jinxiang Chai, Andrew Cove, Alexei Efros, James Hays, Sophie Jörg, Sang Il Park, Paul Reitsma, Moshe Mahler, James McCann, Nancy Pollard, Liu Ren, Alla Safanova, Ronit Slyper, and Jeff Smith.

I would also like to thank all the members of University of Washington's GRAIL, including Steven Seitz, Zoran Popović, Barbara Mones, and Brian Curless, without whom I would never have begun my career in graphics.

I want to thank Anya for her patience and caring in a year of upheaval.

Finally, I would like to thank Pittsburgh for giving me music, love, and laughter.

Abstract

Animation techniques for controlling passive simulation are commonly based on an optimization paradigm: the user provides goals a priori, and sophisticated numerical methods minimize a cost function that represents these goals. Unfortunately, for multibody systems with discontinuous contact events these optimization problems can be highly nontrivial to solve, and many-hour offline optimizations, unintuitive parameters, and convergence failures can frustrate end-users and limit usage. On the other hand, users are quite adaptable, and systems which provide interactive feedback via an intuitive interface can leverage the user's own abilities to quickly produce interesting animations. However, the online computation necessary for interactivity limits scene complexity in practice.

This thesis presents two methods for controlling the rigid body simulations. The first is Many-Worlds Browsing, a method which exploits the speed of multibody simulators to compute numerous simulations in parallel (offline and online), and allow the user to browse and modify them interactively. By bolting responsive, powerful, intuitive interfaces onto relatively simple sampling techniques we get a method that enables animators to produce compelling results with a minimum of effort. The second method is time-reversed simulation: we provide only the final resting configuration of the system and run the simulator backwards in time. During the development of this method we encountered a number of surprisingly counter-intuitive results, which can be elucidated using a combination of numerical simulation and thought experiments.

Contents

List of Figures

Chapter 1

Introduction

The use of physically based simulation has greatly simplified the creation of effects such as water, fire, and interacting solids. Methods developed in computer graphics (alongside other fields such as applied math) have seen ever-increasing levels of sophistication, to the point that companies specializing in film effects routinely simulate oceans [TGP07, KT07], explosions in space [SRF05], melting robots [REN$^+$04], and an enormous variety of other phenomena. It is important, however, to remember that the true consumers of this simulation technology are the animators and technical directors who use the simulation within the context of a shot, and the true measure of a simulation is in how well it tells a story. Often, the most difficult part of physically based modeling is getting the simulation on the computer to match up with the one in the mind's eye of the director.

In recent years, graphics researchers have put an increasing amount of focus on *control*, broadly defined as the problem of making the simulation match the artist's or director's goals. Prior approaches to the problem can be grouped into a few categories: penalty methods, constraint-based methods, and optimization techniques.

1.1 Penalty approaches

The simplest methods for driving a simulation in a particular direction apply forces or impulses directly. These methods can be as simple as attaching a spring to a particle for cloth [BMF03] or rigid bodies or as complicated as matching the current density field of a smoke simulation against a goal key frame and computing a pressure field that impels one toward the other [SY02, FL04]. Due to their simplicity, these were the first techniques developed and are still widely used throughout the effects industry as they are simple to implement, have good run-time performance (even for stiff springs thanks to implicit integrators), and are easy for animators to understand [KANB03]. Penalty approaches are particularly important for clothing [BMF03, BMWG07] as few other techniques have been shown to be particularly suitable.

Penalty approaches have two primary drawbacks. First, they provide no guarantees that they will generate the desired results and as a result significant tuning of parameters is necessary. This is particularly a problem with rigid bodies because collision constraints can render spring and other forces incapable of affecting the simulation (no amount of pushing a block against a surface will allow it to tunnel through). Secondly, strong spring forces tend to produce simulations that appear quite contrived. If the dynamics are active already, as with an articulated character, this may not be a problem. For completely passive systems, however, the animator must trade off the risk of setting the constants too low and not reaching the goal against the risk that she sets the constants too high and produces *implausible* motion. Optimization methods discussed later attempt to capture this contention explicitly.

1.2 Constraint-based methods

An alternate approach is to restrict the domain so that the only simulations possible are ones that satisfy the user's constraints. We can do this by simulating using a basis which only contains states that satisfy user desires [ANSN06] or by using methods developed for solving Differential Algebraic Equations (DAEs) to ensure that the system stays on the constraint manifold [AP98]. Some of these methods also discard physical correctness in exchange for more controllability [BPP01]. The advantage of constraining the solution is that we can put strong guarantees on how the resulting simulation will look and in most cases we do not add associated artificial damping to the system as we might with stiff control springs and implicit integrators.

However, not all user desires can be easily expressed using constraints. Constraints work best for attaching rigid bodies via joints and are commonly used to specify the positions of particular particles [PB88, BW98] or to enforce the inextensibility of links in cloth simulations [GHF+07]. Softer user goals like "land in this spot," however, do not map well onto a constraint-based framework because in this case the constraint formulation requires specifying not just the final frame but the frames leading up to it, an excessive burden on users. Furthermore, these methods ignore the question of plausibility altogether and will apply any force necessary to match the goals; this heavy-handed approach inevitably leads to unrealistic-seeming simulations.

1.3 Optimization

Optimization explicitly addresses the trade-off between physical plausibility and user goals discussed in section 1.1 explicitly. Essentially, the optimization problem boils down to three components,

1. A *means* of control, some parameters that when changed will influence the resulting simulation. These could be forces applied to the system as described in section 1.1, although in most cases they need to be parametrized using some basis to reduce the dimensionality of the system. Alternately, they could include passive parameters of the system such as the initial system state, surface normals, or viscosity.

2. A *metric*, which contains two terms. The first term measures whether our simulation has succeeded in matching the user-specified result, while the second measures plausibility and captures things like the magnitude of the control forces or the deviation of the initial conditions from the user-specified values.

3. An *optimizer*, which minimizes the metric value. Optimizers used in practice include methods like gradient descent which require that the space be continuous and discrete sampling techniques that assume nothing about the space but are much slower to converge. In practice, hybrids of continuous and discrete methods are often used in practice.

Optimization has a number of appealing characteristics. It removes the need for users to spend time adjusting simulation parameters, it automatically accounts for the trade-off between plausibility and user constraints, and (modulo computational constraints) it can guarantee that, within the confines we have set, the solution returned by the optimizer is the best that can be found.

Unfortunately, optimization has a number of drawbacks that limit its applicability in the effects industry. In high dimensional spaces (e.g. the space of possible simulation parameters), optimizers either take a long time to converge or else return suboptimal solutions, which can make a substantial difference in the quality of results [SH07]. Techniques that rely on gradients converge significantly faster [PSE$^+$00, PSE03] but are highly subject to local minima and require that the simulator be differentiable, which is frequently not the case for collision handling [Bar94]. With or without gradients, the inner loop of the optimizer is necessarily at least as costly as the simulation itself, which limits applicability to problems of smaller size. Those experienced with optimization are also familiar with the amount of care and attention that is required to get convergence to a high-quality solution, especially in many-dimensional space about which we have little intuition.

But perhaps the most problematic part of optimization is the work flow. Long optimizer convergence times mean that the *artist* will get significantly fewer iterations on a particular problem. The result is that she must have a firm idea in advance about how the result should look and provide sufficient specificity to the optimizer lest it have the freedom to return the wrong result after wasting hours of computer time. And yet providing too *many* key frames can produce poor quality results as well, as users may lack good intuition about the timing of the physics and hence specify constraints that are nearly impossible to satisfy without huge artificial forces. The ultimate result of this is that users invariably find themselves iterating on the constraints and other optimization parameters — much as they might have iterated on spring constants in section 1.1, but with a substantially higher turnaround time (although, to be fair, much of the work is happening in the background, which may reduce labor costs).

Nonetheless, gradient-descent optimization has been applied in recent years with considerable success in the areas of fluid animation [TMPS03] and cloth [BTH$^+$03, WMT06]. Much of this work focuses on the need to efficiently compute gradients [MTPS04]. Unfortunately, gradient descent struggles with the bifurcations in the parameter space that result from collision handling [WMT06]; this makes even the most sophisticated methods developed for the optimization of fluids unsuitable for rigid bodies. Gradient-based methods such as those developed by Popović and colleagues [PSE$^+$00] either resort to random sampling when discontinuities are encountered or rely on a good initial guess [PSE03] to ensure convergence.

1.4 Many-Worlds Browsing

While there has been little follow-up work in controlling the passive dynamics of rigid-body simulations, great strides have been made in pure simulation of multibody dynamics. In addition to the impressive results seen in several recent academic papers [GBF03, KEP05, WTF06], rigid dynamics have become ubiquitous in video games. A number of fast, stable solvers are available both commercially (e.g., Havok, Ageia) and as open source (e.g., Open Dynamics Engine, Bullet Physics Library). Even more recently, we have begun to see the hardware acceleration of rigid dynamics, producing impressive real-time demonstrations involving thousands of interacting bodies on both the Havok FX and Ageia PhysX platforms. Many physics algorithms, particularly collision detection, are trivially parallelizable and map easily onto existing graphics hardware, and the recent acquisitions of Ageia by NVIDIA [Hru08] and Havok by Intel [Sto07] and the increasingly widespread adoption of highly parallel stream processing architectures such as Sony's Cell and Intel's Larrabee suggest that the trend of offloading physics onto relatively special-purpose machinery is likely to continue.

We naturally seek to leverage this technology to accelerate the problem of controlling rigid bodies. Unfortunately, gradient-based techniques do not parallelize well: fast algorithms for gradient computation (e.g. the adjoint method) have an explicit ordering which limits parallelization, and it is not possible to compute the next gradient until after the current gradient step has been taken. Sampling techniques, on the other hand, are embarrassingly parallel. Sampling also has advantages over continuous optimization for larger scenes; increasing the number of collisions affects the cost of gradient computation for continuous techniques [PSE$^+$00] and also boosts the number of discontinuities and local minima in the space. This is a problem for offline techniques, which quickly slow to a halt, but also degrades the usability of online techniques as very small changes at one point of the simulation can drastically affect the remainder. On the other hand, sampling methods may in some cases function *better* in complicated, collision-prone situations, because each collision event adds more parameters to affect and thus increases the amount of variability in the data set.

Unfortunately, in these high-dimensional spaces, naïve sampling fails to find solutions to all but the most simple constraint problems. Chenney and Forsyth [CF00] demonstrated that it is possible to address this through the use of more sophisticated sampling algorithms, but these algorithms require a substantial amount of per-problem tuning and user expertise to ensure convergence. Perhaps most problematic, however, is that the Markov Chain Monte Carlo methods used in that work took many hours to converge, which means that all the difficulties related to the use of optimization in the production pipeline discussed in section 1.3 apply.

Much of the poor performance of MCMC can be attributed to the fact that the proposal distribution must be specified before optimization can begin, which besides being difficult for untrained users means that if the user's intuition proves incorrect convergence will be very slow due to poor mixing. The Markov condition which states that the system cannot retain any memory of prior states ensures that the method cannot reuse successful strategies or avoid problematic ones. Memory-less sampling is used in statistics because it ensures ergodicity, an important property to have when computing a statistic of the distribution (e.g., the mean). When trying to compute the maximimum of a function, however, the Markov condition may be counterproductive.

On the other hand, if we eliminate the Markov condition and try to explicitly learn the mapping from control parameters to the output space, we will struggle with the high dimensionality and non-linearity of the mapping. Discontinuities in the space only increase the difficulty as learning techniques typically assume some variant of Lipschitz continuity. Given this, computing solutions to complicated constraint problems might seem impossible – and yet, the success of simulation in film production suggests that artists have been able to find solutions to these problems. Users have a number of advantages, however. Many optimization problems are over-specified, as languages for describing problems may lack the nuances to contrast "must have" and "should have" with "would be nice to have." Users interacting with the system can determine relatively quickly whether the problem has been rendered impossible to solve by, for example, poorly placed collision geometry.

Hence, it seems useful to keep the user in the loop, helping to guide the optimization. This has a number of benefits. The user can avoid spending time tuning the optimization process itself, which is time-consuming and error-prone because useful feedback is generally nonexistent. Certain kinds of constraints, e.g. "look exciting" or "land right-side up," can be difficult to specify precisely and are likely to be neglected in the initial optimization round, leading to results that are technically correct but lack the desired "punch." Furthermore, keeping the user in the loop enables him to respond quickly if preliminary results reveal that his pre-specified constraints are not going to produce the desired animation, rather than require him to await the result of a long optimization process. This is especially important during the process of pre-visualization, where rapid feedback is more important than guaranteeing that the resulting simulation be optimal with respect to any particular metric.

We therefore advocate a *user-driven* alternative to optimization for solving multibody constraint problems. We will perform sampling in parallel on a cluster of machines to keep the entire approach interactive, which will give the user immediate feedback on existing possibilities. Because this will generate a substantial amount of data, we need to give the user tools to steadily guide the sampling toward the result she seeks. Our interface will show the user the paths taken by the bodies, and the user can sort through them and inform the system which parts of the produced motion to keep and which to discard. We can then generate new simulations which only modify the parts of the motion that the user requested, but still guarantee physical plausibility.

To make this system feasible, the examples must be stored on the user's machine and the user must be able to easily sort through the hundreds of example motions without having to examine each one individually. For inspiration, we look to the successful interfaces used for web search, which take the enormous amount of data present in the World Wide Web and make it palatable. Classically, web search results were constrained by Boolean combinations of keywords; results which satisfied these constraints were ordered by some ranking mechanism which tried to mea-

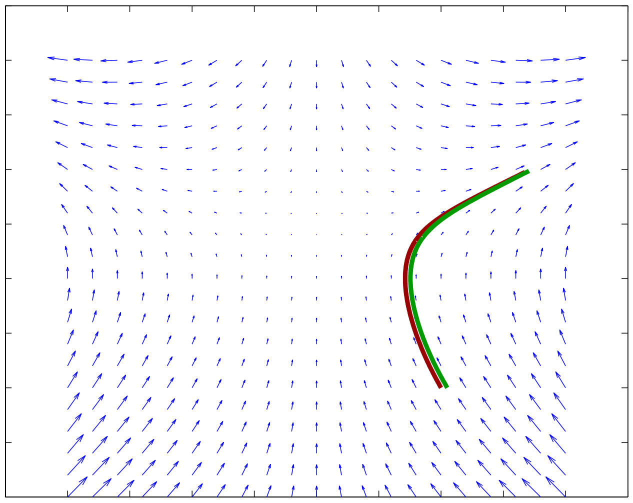


Figure 1.1: *For* **non-dissipative systems***, reversing time is trivial; in this first-order example we use Euler's method to integrate a particle through a force field, with* $\frac{dx}{dt} = 2xy$ *and* $\frac{dy}{dt} = x^2 - 2y + 1$*. Small perturbations in the initial conditions for the time-reversed problem will produce only small perturbations in the final result, which means that we do not need a particularly good estimate of the final state to infer what the initial state could have been.*

sure both the relevance and the authority of the returned pages [Mau97, PBMW98]. Our system has two similar components: screen-space queries, which allow the user to quickly limit the set of results returned, and metrics, which allow users to choose the "best" among the remaining examples.

Cluster-based sampling assures interactivity for moderately-sized problems. In the current system, larger systems involving hundreds of rigid bodies can be computed offline, and thanks to in-core compression and storage of the example simulations the user still able to browse through the results interactively. Unfortunately, the relatively high cost of performing refinement on these examples limits the ability to solve complicated boundary-value problems on these larger examples (e.g., with both start and end states specified). We hope that this will be addressed by both faster simulations (our current implementation does not use hardware acceleration) and better algorithms.

1.5 Time-reversed simulation

One special case that we may often wish to solve involves problems where only the end state is specified. This can be useful for many of the types of simulations that we see in movies; shorter shot lengths mean we may cut from the start of a simulation to the final few seconds as the objects come to rest, ignoring the dynamics in between. We can solve these problems by running the simulation *backward*, instead of forward. That is, the user specifies the final condition of the system (possibly at rest) and step it *backward in time* until we have generated enough motion.

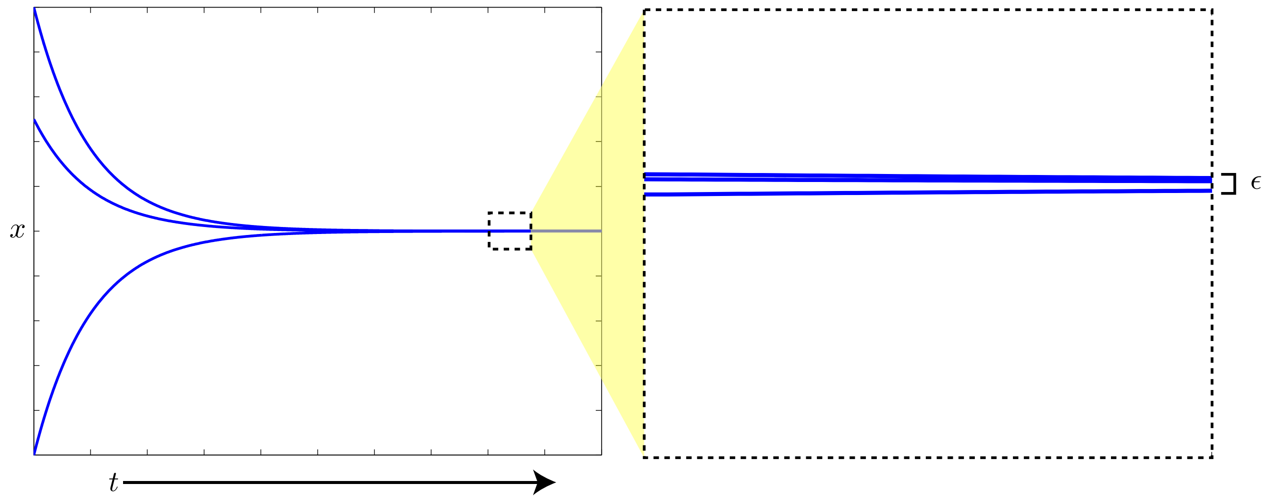


Figure 1.2: *For* **dissipative systems***, very small perturbations in the final resting state can produce massive changes when the dynamics are run backward. Here, we plot three distinct solutions to the first-order system $\frac{dx}{dt} = -1$, each starting with a different initial value (left). If we examine the three states at time $t' \gg 0$, we can see that they are separated only by a very small value ϵ. This means that if we run the system backwards from time t', the selection of initial state will have a profound effect on the resulting dynamics.*

In simple mathematical terms, one can consider "time reversal" of Newton's equations of motion, $\mathbf{f} = \mathbf{Ma}$, as just the replacement of time, t, by $-\tau$, where τ is a "reverse time" variable:

$$\mathbf{M}\frac{d^2\mathbf{x}}{dt^2} = \mathbf{f}\left(\mathbf{x}, \frac{d\mathbf{x}}{dt}, t\right) \quad \xrightarrow{t \to -\tau} \quad \mathbf{M}\frac{d^2\mathbf{x}}{d\tau^2} = \mathbf{f}\left(\mathbf{x}, -\frac{d\mathbf{x}}{d\tau}, -\tau\right). \tag{1.1}$$

If the forces are autonomous and independent of velocity, e.g., $\mathbf{f} = \mathbf{f}(\mathbf{x})$ (see Figure 1.1), then the forces will look the same forward or backward in time, and the system can be said to have *time-reversal symmetry* [Rei99]. However, for other systems, such as those involving velocity-dependent damping forces, the dynamics are not necessarily reversible: if energy is lost moving forward in time, then it is gained moving backward in time (see Figure 1.2). While the extension to rigid body animation might seem straightforward, unfortunately, a number of difficulties will arise related to our reversal of causality; while we offer partial solutions to many of these issues, there is much room for future work here.

Perhaps the most discouraging fact is that, unlike in forward simulation, we can not uniquely solve a backward simulation problem, in general. For dissipative systems, such as rigid bodies with frictional contact, the problem of generating motion ending in a given state is vastly underdetermined–mathematically speaking, it is an ill-posed problem. For example, take a single block sitting at rest on a plane, and consider the nearly limitless set of motions ending in that position (see Figure 1.3). It could have been sitting in place for a second or a year. It could have balanced on a single edge for an arbitrary length of time before falling victim to gravity and landing on its face. It could have slid in from any compass direction, while rotating (or not) about its vertical axis. It

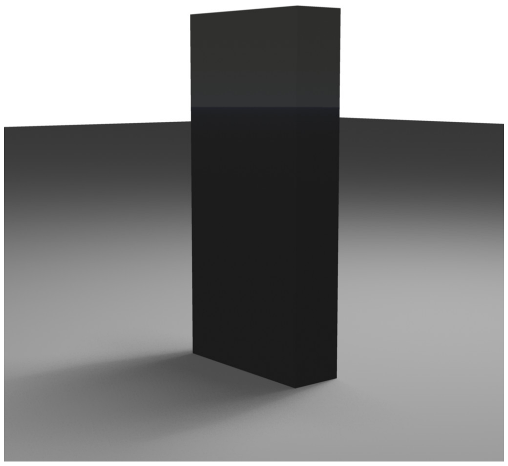


Figure 1.3: How did this block come to rest? *We can not know, but we can generate plausible candidate motions using time-reversed simulation.*

could have bounced some before sliding, or simply fallen a vertical mile only to land perfectly in place. With friction, it is even possible for the box to slide, then bounce, then slide again. If the geometry were rounded, it might have rolled into place, perhaps while sliding or bouncing.

In Chapter 5, we will consider time-reversed simulation of rigid bodies with frictional contact. Although we cannot find a unique solution to a given end state, this is not necessary within the Many-Worlds Browsing framework. Instead, we can sample *possible* backward motions that could have generated the final simulation state. Doing so will require a holistic approach: we must consider not just the mathematics and time-stepping scheme, but also sampling strategies and ways to communicate user intent about the desired motion, e.g., "when should it start moving, and in what way." We will start by considering a mathematical framework for backward simulation, which will entail changes to the Linear Complementarity Formulation to enable it to step backward in the presence of frictional contacts.

Unfortunately, merely being able to take backward time steps of some *possible* motion does not guarantee that we will generate long sequences of *plausible* motion. The true challenges and limitations of backward simulation emerge only *after* we have designed and implemented a time-stepping scheme. It turns out that the naïve application of reverse simulation results in motion that, while perhaps technically and physically possible, can be qualitatively and quantitatively different from typical forward simulations. Therefore, we will catalogue a number of situations where any basic reverse simulator with a limited time horizon will generate noticeable artifacts. Where appropriate, we also suggest possible solutions, which may in some cases violate physics, but which produce noticeably less offensive motion.

Introducing backward simulation into the existing Many-Worlds Browsing framework expands somewhat the space of possibilities. In particular, we can run some scene objects forwards in time and others backwards in time, producing highly improbable and entertaining effects. We can also add objects into the scene midway through the simulation and run them both forward and backward, ensuring correct interactions with other objects throughout.

Chapter 2

Related work

2.1 Rigid body control

Control techniques were first applied to rigid bodies through actuated joints in the context of character animation, such as in Spacetime Optimization [WK88]. Tang and colleagues [TNM95] extended these techniques to passive simulation in the context of a frictionless two-dimensional rigid body simulator [TNM95]. Using a genetic algorithm, they were able to find solutions to problems where the initial and end states for up to 14 discs were specified but initial velocities were allowed to vary. However, their algorithm assumes that the motions of the bodies are mostly independent, and it is not clear how well it would extend to situations with friction and more interaction between objects.

Varying only the initial conditions of a body does not generally allow enough control to handle complicated boundary value problems. Barzel and colleagues [BHW96] increase the amount of possible control through the concept of "plausible simulation." They describe how rigid body simulations can be modified by slightly perturbing collision normals. The primary focus of the paper is the use of this manipulation to reduce the "sterile" appearance of rigid body simulations of perfect (Platonic) solids. They do, however, suggest three different possible interfaces for controlling the motion: one for building motion paths segment by segment, one for interactively manipulating trajectories, and one for finding solutions to pre-specified boundary value problems where start and end positions are given. However, the authors do not explain how to implement either the first or the second interfaces, but instead focus on the third. As an example of a boundary value problem, they present a simple example involving pool balls. To find a possible solution, they trace paths backwards from the prescribed end state, keeping track of the set of possible incoming paths in a sort of motion "cone." Once this cone intersects the specified start positions, a solution is found. However, for 3-dimensional interacting objects the set of plausible paths is not going to be a simple cone; it will be instead a complicated structure (embedded in a $6n$-dimension space) that is not generally convex, which makes this simple approach impractical.

In general, rigid body simulation is highly nonlinear and discontinuous, due to complicated friction models and binary events (for example, the difference between two objects colliding or not). Thus, for some problems, stochastic approaches have been most successful. Chenney and Forsyth used the Markov Chain Monte Carlo algorithm (MCMC) to sample possible solutions to constraint problems [CF00]. While their approach was successful in finding solutions to a variety of problems, it took many hours to run. Furthermore, the MCMC algorithm is notoriously difficult to tune as good results depend on the Markov chain having rapid mixing [Was04]; in the case of Chenney and Forsyth, this meant that a careful selection of the proposal distribution was necessary for each example. It is probably unreasonable to expect animators to be capable of choosing a good distribution; hence, the technique would seem to be impractical for use in the production pipeline, or at least result in more parameter tweaking.

Offline optimization approaches that require the user to specify all of the constraints and objectives beforehand suffer from the problem that the user seldom knows exactly what he wants *a priori*, and even if he does he may under-specify the problem so that the optimizer gives a result that is technically correct but not what was desired. If the optimizer must run for many hours before returning an answer, the result will be much wasted production time. It may then be better to provide the user with a less sophisticated optimizer that she can guide toward the solution interactively than a more sophisticated optimizer that always returns the “correct” answer, but after many hours of computation. Examples of this include work by Cohen [Coh92], where even a fairly non-intuitive UI improves the results over previous spacetime optimization approaches. Lazlo and colleagues simply gave their users a set of keyboard controls with which to manipulate the muscles of a physically based character and let the user learn the most efficient way to control the character [LvdPF00].

Popović and colleagues [PSE$^+$00] presented a particularly compelling interface in which the user could interact directly with the simulation to produce desired results. In their system, the user can select an object at any point during the simulation and manipulate it. A rapid gradient descent algorithm is combined with some limited random sampling where necessary to find solutions that satisfy user constraints. If no such solution can be found at interactive rates, the user is notified immediately and can attempt to apply different constraints. The primary drawback of this approach is that it cannot be applied to very large systems, as gradient computation is linear in the number of collisions and complicated scenes may involve thousands of collisions. It is also tied to the particular simulator used; in particular, it is unclear how to accommodate non-differentiable friction models in common use [Bar94].

In their followup paper, Popović and colleagues introduce the use of multiple shooting to enable them to solve significantly harder optimization problems [PSE03]. This, however, turns the optimization into an offline problem (5-20 minute optimization times); they address the loss of the interactive interface by introducing a novel sketching interface: a rigid body is instrumented and the user moves it in approximately the desired fashion. The algorithm is then able to recover realistic timing for the final motion. This form of sketching is a compelling interface for simulations involving small numbers of objects, but is impractical for simulations involving hundreds of interacting articulated objects.

2.2 Control in other domains

A number of papers deal with the control of smoke and liquid animations involving thousands of degrees of freedom. Early fluid control consisted of either the application of force fields (or equivalently, pressure gradients), or direct modification of the velocity field. Foster and Metaxas defined a variety of controllers of both kinds, which can be applied at runtime to modify the fluid flow [FM97]. A followup paper [FF01] described a more intuitive interface for a direct velocity controller using space curves. Beaudoin and colleagues [BPP01] presented a technique for describing the velocity field for a simple fire simulator. These techniques work well for situations where the fluid motion is primarily the result of some large external force, as for a fountain or an explosion. They are less directly applicable in the case of largely passive fluid behavior. Direct velocity control has also been applied to create virtual fluid characters, such as the melting Terminator character [REN$^+$04]. Here, this kind of direct control makes sense because a fluid *character* must match the keyframed motion very closely; communication of story and emotion to the moviegoer is far more important than other considerations such as minimizing control forces.

In their 2003 paper, Treuille and colleagues introduced the idea of keyframing fluid animations [TMPS03]. In essence, instead of having the user control fluid motion through the use of velocity or force fields, they instead allow the user to directly specify the rendered result at several points in time (the key frames). This may consist of a smoke density field [TMPS03] or a fluid volume [MTPS04]. They combine multiple shooting techniques similar to those used by Popović and colleagues [PSE03] with a method for differentiating through a voxel-based smoke simulation. A followup paper extends the method to three dimensions through the use of the adjoint method which reduced the computational complexity of derivative calculation [MTPS04] and adds support for free surfaces such as water. These methods have limited applicability for multibody dynamics due to the large numbers of discontinuities that occur at collision events. Moreover, unlike smoke control, where increased control forces can always move free-space smoke into desired configurations (albeit less subtly), increased control forces in multibody dynamics cannot ignore configuration space obstacles, e.g., slamming two objects together harder will not make them pass through.

For fluid simulations where the primary goal is hitting target key frames, a global optimization is often unnecessary. It is sufficient instead to apply an small external force at each time step that gradually guides the fluid toward the provided target shape. The advantage of this approach is that its computational cost is within a constant factor of the original simulation, whereas even the most sophisticated optimization approaches take much longer. The trade-off is the loss of any guarantees as to the optimality of solutions with respect to, e.g., the amount of control forces used. It is also harder with these methods to manage trade-off between accuracy in hitting the key frames and the amount of control used to do so. Yu and Shi [SY02] proposed this idea first and were able to demonstrate it matching both 2D and 3D shapes, include 2D movies. They followed this up with a more sophisticated version [SY05a] that uses the level set method to reduce artifacts, and later extended it to free surfaces as well [SY05b]. Fattal and Lischinski [FL04] applied a similar idea for guiding smoke toward desired key frames and added an additional term which tends to oppose diffusion. Using this extra term they are able to hit key frames quite exactly.

Some methods for generating 3D fluid animation start with a collection of examples of 2D motion and use a subset of these to generate the final 3D motion [RNGF03, MMS04]; however, these methods do not address the problem of how to assist the user in selecting which examples to use. Pighin and colleagues [PCS04] convert an Eulerian fluid simulation into a set of path lines, which can then be selected and edited. There is no guarantee that the resulting edit will be physically plausible.

In some cases, we can significantly increase the controllability of a simulation by using a different simulation technique. This is especially true for fluid simulations; choosing the right basis for the velocity field can allow users to "design" the desired motion field. Lamorlette and Foster [LF02] use spline curves as primitives for their fire simulator and model the actual fire as a volumetric field around these curves. This is used with great effect for a fantastical dragon's fiery breath. Angelidis and colleagues [ANSN06] describe fluid motion using closed curves called "motion filaments." The use of this basis combined with user-provided filaments provides a level of control similar to the direct velocity manipulation techniques above, but with a more intuitive interface. Schpok and colleagues [SDE05] extend these techniques to Eulerian grid simulations by decomposing the velocity field into vortex and laminar features, which can then be edited to produce the desired results. Generally, these kinds of simulation techniques are more applicable to smoke than to fluids with free surfaces, as direct velocity manipulation is more noticeable in that domain and bases such as spline curves or vortex filaments generalize less well to surfaces.

One alternative interface to the voxel keyframe approaches is provided by Thürey and colleagues, who use *control particles* to direct the flow [TKPR06]. Fluid is pulled toward the control particles, which can be generated either using another fluid simulation (they demonstrate running a simulation backward), a mesh animation (as in previous approaches) or through user specification. By controlling only the low-frequency detail and allowing the fluid simulation to provide higher frequency detail, they are able to get compelling animations without the need for voxelized keyframes. This idea was applied to cloth by Bergou and colleagues [BMWG07], and seems to produce particularly compelling examples there; this may be because when controlling cloth we frequently seek to get the large-scale behaviors correct but have little interest in sculpting particular wrinkles.

Wojtan and colleagues suggest a more traditional optimization-based approach to controlling cloth that uses a novel variant on the adjoint method to compute gradients for implicit deformable simulations [WMT06]. An alternative approach to cloth control is to try to match the "feel" of a particular piece of cloth, which generally amounts to matching the parameters on a cloth simulation that may or may not be physically accurate. Breen and colleagues pioneered the use of the results of the Kawabata test for computing the static parameters of cloth [BHW94] and Bhat and colleagues extended this to the dynamic parameters through a vision-based interface [BTH$^+$03].

Much of the work controlling deformable simulations has focused on direct control through the use of springs or constraints. By controlling certain degrees of freedom through these methods but still applying equations of motion, it is possible to get realistic physical effects such as elasticity and secondary motion. For example, Platt and Badler [PB81] drove a facial model using contracting "muscle fibers." Platt and Barr introduced a technique for applying constraints to the vertices of deformable objects using Lagrangian multipliers [PB88]. In a similar vein, Metaxas and Terzopoulos control a simple snowman character to follow a pre-specified path [MT92]. Terzopolous and Witkin used similar controls to animate deformable vegetables [TW88]. Kondo and colleagues introduce a technique for the control of elastic objects [KKiA05] in which users can set direct key frames on either positions or velocities. This is able to produce realistic-looking deformations that satisfy user constraints. For all these methods, constraints are exact, so if the user chooses poor key frames the resulting motion is physically implausible. Thus, these methods work well when the user knows what paths objects should follow, but would be impractical for simulations containing hundreds of objects.

In the area of character animation, there are a number of techniques giving animators intuitive controls for producing realistic body deformations. Several techniques [WG97, CZ92, TBHF03] drive physical muscle models to get realistic muscle deformation in response to character movement. Capell and colleagues show how to drive an elastically deformable character using an underlying skeleton model [CGC+02], and Sifakis and colleagues [SNF05] drive a deformable face model using data from motion capture. This kind of direct control is primarily useful in character animation, where users require very precise control over the character's movements.

Kircher and Garland introduced a technique for editing deformable animations [KG06]. They use this to produce examples such as a face appearing in a waving cloth motion. Singh and Fiume [SF98] use a physical process to animate space curves, which then deforms a curtain in a physically plausible manner. Using these curves, they then edit the motion to make the curtain form a face. These techniques do not ensure that the resulting edits are physically realistic, however.

Tu and Terzopoulos [TT94] used simulated annealing to learn control of a deformable fish model. Grzeszczuk and colleagues [GTH98] were able to accelerate this using function learning on neural networks. While local control may be a good way to get virtual characters to perform in a desired fashion, it is not effective for getting a particular global result in the presence of large number of collisions.

2.3 Browsing simulations

Our work on user interfaces for selecting among the computed examples is most closely related to Design Galleries [MAB+97]. In this approach, examples are arranged in a two-dimensional layout using an algorithm which attempts to maintain distances in the low-dimensional embedding. The approach is demonstrated on a simple particle system, a 2D double pendulum, and a controlled 24-DOF "hopper dog." Chenney and Forsyth [CF00] suggest that such a system could be used to browse examples of rigid body motion generated using their sampling approach. For multibody dynamics involving dozens or even hundreds of bodies, however, to try to capture all the complicated inter-body interactions in a single 2D arrangement would not be feasible. Here we again draw on the Google analogy; imagine finding a 2D layout for even a small portion of the web and then asking a user to locate the ACM SIGGRAPH homepage. Ever-changing user requirements make the task even more difficult, because "similarity" between two scenes will be dependent on, e.g., which part of the scene the user is focusing on.

The idea of bringing the user into the optimization process has been exploited by Cohen [Coh92], who demonstrated that even a relatively unintuitive interface can, if it provides good feedback, leverage human capabilities to improve the results of spacetime optimization. Laszlo and colleagues [LvdPF00] give their users an even simpler set of basic keyboard controls, to which they rapidly adapt and are soon able to control a physically based character without the need for an optimizer.

Computational steering [vLMvW96] is an emerging subfield of scientific computing which incorporates visualization and interactive feedback to adjust long-running simulations. In a steering environment, researchers can view the current state of the simulation (which is usually running on some high-performance, remote cluster) and adjust parameters if necessary. Unlike our technique, however, users must adjust parameters to change the output, whereas our approach allows selecting among different outputs directly. Many of the same issues we encounter, such as dealing with large amounts of data, come up in this and other areas of scientific visualization.

2.4 Animation compression

There is an extensive literature on the topic of compressing time-dependent geometry [Len99, AM00a, BSM$^+$03, IR03, GK04b, SSK05]. However, the main focus of most work is on compressing *mesh animations*, rather than animations of rigid bodies. Many papers [Len99, AM00a, JT05] use affine or rigid transformations to help with compression; however, because the storage space required by a 4x4 transformation matrix is much less than the mesh itself, no effort is generally made to compress these transforms. Arikan obtained 30:1 compression on motion capture data [Ari06], which consists of translational and rotational components for each limb; however, he was able to achieve this using coherence between joints, which we cannot use for rigid body compression.

There is a substantial body of work concerned with the compression of time series data. The simplest such method is Differential Pulse Code Modulation (DPCM), which is often used in audio compression because it can be run easily in real time. In DPCM, each sample X_i is predicted using the k previous samples $X_{i-1}, X_{i-2}, \ldots, X_{i-k}$. The difference can be encoded using fewer bits than the original signal [Sal04]. A similar idea is used in the SHORTEN codec [Rob94]. However, this technique is ill-suited for our purposes because the number of bits required depends linearly on the frame rate of the animation, and we would prefer to keep our data at a relatively high frame rate for computing accurate motion blur. We do use this scheme for compressing spline coefficients, however, due to temporal coherence between them.

Our desire to retain a high frame rate suggests that we should look at function space approaches to compression. These can be broken into two major groups: frequency space methods and wavelet techniques. Frequency space techniques are commonly used in audio compression because they map well to our own perception of sound. MPEG-1 Audio [BS94] and Ogg Vorbis [Ogg] are two popular audio codecs that use the Modified Discrete Cosine Transform (MDCT). However, frequency space methods will not work well for compressing motion paths because it will take a significant number of high-frequency components to capture first derivative discontinuities due to collisions (for example, consider the Fourier series of a sawtooth wave).

Wavelets offer an alternative to purely frequency-space methods which are localized in space. Wavelets are most commonly used for image and video compression, but there is no reason they cannot be applied to time series data as well [CF99]. Wannamaker and Vrscay applied a combination of wavelet and fractal techniques to audio compression with some success, although they were unable to beat state-of-the-art frequency space techniques [WV97]. A number of papers have applied wavelets to ECG data [CGWS92, Hil97]; 8:1 compression seems to produce acceptable results in this domain. The major drawback of using wavelet approaches for our motion paths is that for optimal compression the edges of the regions of support of the wavelets should line up with the first derivative discontinuities in the motion, whereas most wavelet schemes used in compression assume the individual wavelet domains are fixed (among other things, this enables simple compression of the wavelet coefficients using run-length encoding).

Piecewise polynomials are not as common for real-world applications as other function space approaches; however, there are some areas where these techniques work well. Prandoni and colleagues [PV99] describe a scheme for fitting piecewise polynomials to data that is in many ways similar to our own; however, they only provide rate/distortion curves for synthetic data, making comparison difficult without running further tests. Nygaard and Haugland [NH98] propose an algorithm that provides an optimal fit (under an L^2 norm) using piecewise polynomials. The technique is based on a dynamic programming algorithm described by Haugland and colleagues [HHH97]. While it would be interesting to compare our polynomial fit to the optimal one for evaluation purposes, the cost of actually computing this for each compressed path would be too high for our purposes.

A number of techniques for compressing time series data focus on capturing important features such as maxima and minima [LKK01, PF02]. This provides a very high compression ratio and can be used to accelerate certain kinds of queries on the stored data; however, quality is insufficient for our purposes. An additional compression technique that is occasionally used for time series data is fit an autoregressive model to the data [NC93]; however, this only works for stationary processes.

2.5 Backward simulation

Rigid body simulation has a rich history in both the mechanics and computer graphics literature. Perhaps the simplest techniques to implement are those which handle contacts by applying instantaneous impulses to the bodies; continuous contacts are handled using a series of small impulses [Hah88, MC95]. While these methods are fast, they tend to handle stacking behavior poorly and do not accurately model friction.

Also easy to implement are penalty-based contact methods, such as Hertz, Kelvin-Voigt, or Hunt-Crossley models (see [HC75, LN94]). As in (1.1), these force-based models are trivial to reverse (just negate the damping term), but can be slow compared to competing constraint-based methods, and they generally do not handle complicated stacking behavior well due to the difficulty in incorporating a proper Coulomb-like friction model (although see [Mir00]).

Guendelman and colleagues introduced a timestepping scheme which interleaves contact handling and timestepping [GBF03]. Contacts are handled one at a time in a relaxation scheme, but convergence is accelerated through use of the contact graph [Hah88]. Weinstein and colleagues extended this method to articulated bodies with control [WTF06]. Kaufman and colleagues [KEP05] solved for the contact forces on each body separately, holding other bodies in the scene fixed. Milenkovic and Schmidl [MS01] formulated the contact problem as a quadratic program. It is likely that time-reversed simulation methods could be developed based on any of these methods, although the issues that we discuss in section refsec:issues would arise regardless of the underlying time-stepping scheme.

Adjoint methods for gradient computation step through the simulation forward and then backward [WMT06, MTPS04], but the backward stage involves performing computations on the existing simulation rather than computing new time-reversed motion. Similarly, time-warp rigid body simulation [Mir00] involves "rolling back" bodies to previously states, but only to states that were previously computed using forward simulation.

Geometric integrators are time-stepping schemes that preserve certain invariants of the system, such as energy [KYT$^+$06, LR05]. These methods are ideal for simulating N-body systems where exact energy preservation is needed to prevent orbits from spiraling inward, and they guarantee that the dynamics are reversible. However, this energy preservation and reversibility does not hold across collision events, and indeed there are cases (e.g. a body at rest) where the trivial solution computed by a reversible integrator will not generate the desired behavior.

Bidirectional simulations are an important research topic in some areas, such as computing the motions of planets, where it is important that the system energy remain constant over long periods of time to prevent orbits from decaying (or exploding). Much of the work there focuses on symplectic integration for Hamiltonian systems, and a comprehensive overview can be found in Leimkuhler and Reich [LR04]. Symplectic integration is a potentially rich source of possible algorithms; however, rigid body motion is only Hamiltonian *between* collisions, so these algorithms do not apply directly to the problems we'll be looking at.

An alternative method that is sometimes used for time-reversed integration is extrapolation. Gear and Kevrekidis [GK04a] demonstrate a simple method in which they take several forward steps, fit an interpolant, and use this to compute a single reverse step. Any technique such as this that assumes that the space is continuous will likely fail in the presence of discontinuous rigid body collisions. Using some knowledge of the problem domain, however, we can develop a reverse solver that is no slower than forward simulation.

Chapter 3

Review of rigid body simulation

While the details of Many-Worlds Browsing are largely independent of the simulator used, our approach to backward simulation will require a familiarity with the details of the linear complementarity problem formulation of rigid body simulation. We therefore provide a review of the concepts here. In principle it will be sufficient to understand only the material through §3.6 to understand Chapter 5. However, some of the approximations commonly used in implementation have a direct bearing on backward simulation and so are discussed in §3.7 in the hope that they will be useful to implementers.

3.1 Contact handling

Techniques for rigid body simulation must (broadly speaking) deal with three main problems: contact handling, other constraints (including controls/motors at joints), and integration. Although we will touch on it briefly in §3.3, for further discussion of the third problem we defer to Baraff's course notes [Bar01]. More information on accurately timestepping rigid bodies can be found in work from the mechanics literature [SW91, PC93, KE05]. Constraints are handled in our simulator using methods identical to the ones described in Witkin's course notes [Wit01], although other methods are available [Bar96, WTF06]. As contact handling will be the major focus of our backward simulation algorithm, we will review the Linear Complementarity Problem (LCP) formulation here. This formulation has been extremely well-studied and is implemented with the Open Dynamics Engine (ODE) from whose codebase our solver derives.

As is common in rigid body work, we will assume the existence of collision detection code which will compute a sufficient set of contacts to prevent interpenetration. Baraff discusses the problem in some detail in both course notes [Bar01] and papers [Bar94], and there is a huge body of work in this area [Mir98, GLM96, LA06, BO04].

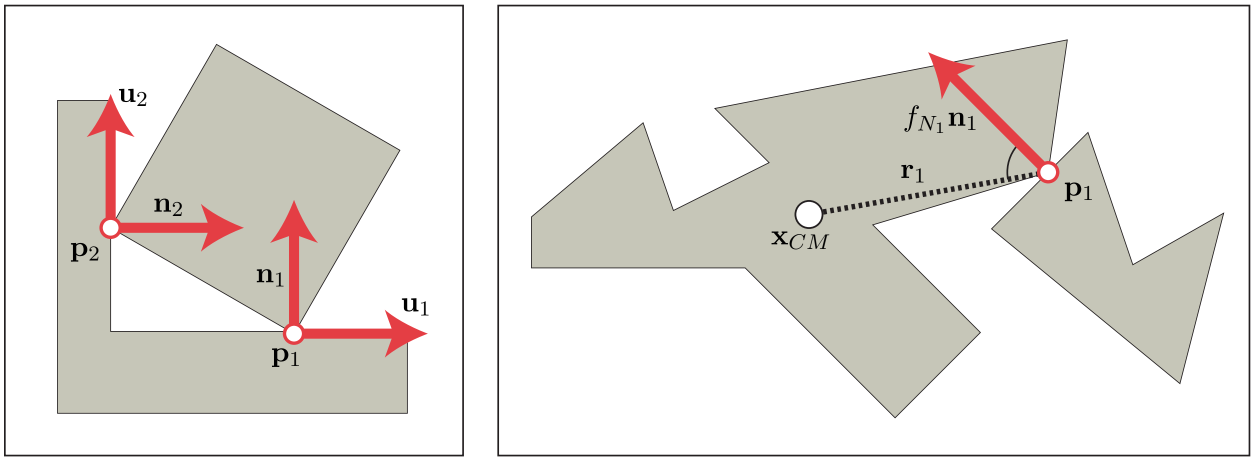


Figure 3.1: Left: *A contact point i consists of a point $\mathbf{p}_i$ and a normal $\mathbf{n}_i$. If the relative velocity and acceleration of the two objects at $\mathbf{p}_i$ projected onto the $\mathbf{n}_i$ is 0, the two bodies are guaranteed not to interpenetrate in the next timestep. To handle frictional contact we add an additional pair of vectors $\mathbf{u}_i$ (pictured) and $\mathbf{w}_i$ (which in this image would be pointing out of the page).* **Right:** *Regardless of how complicated contact geometry is, the relationship between force and torque is simple; here, $\mathbf{f} = f_{N_1}\mathbf{n}_1$ and $\tau = \mathbf{r}_1 \times (f_{N_1}\mathbf{n}_1)$.*

3.2 Frictionless contact

Each contact consists of a point $\mathbf{p}_i$ and a normal $\mathbf{n}_i$ (Figure 3.2, left). We assume our collision detection code returns a set of contact points $\mathbf{p}_1, \ldots \mathbf{p}_n$. The scalar normal velocity $v^t_{N_i}$ evaluated at time t is the relative velocity of the two bodies at $\mathbf{p}_i$ along $\mathbf{n}_i$, and $a^t_{N_i}$ is the corresponding normal acceleration. The convention is that if both the normal velocity $v^t_{N_i}$ and acceleration $a^t_{N_i}$ at each contact point are non-negative, then the two bodies will not interpenetrate in the next timestep (although in the absence of continuous collision detection, interpenetration may occur between collision checks). To maintain this invariant, we are allowed to apply a normal force $f_{N_i}\mathbf{n}_i$ at each contact point, subject to constraints that will be discussed in detail in §3.4.

Given the point $\mathbf{p}_i$ and a force $f_{N_i}\mathbf{n}_i$ applied at that point it is easy to compute the force and torque acting on the body,

$$\begin{aligned} \tau &= f_{N_i}(\mathbf{r}_i \times \mathbf{n}_i) && (3.1) \\ \mathbf{f} &= f_{N_i}\mathbf{n}_i && (3.2) \end{aligned}$$

As forces and torques add linearly, if we have many contacts, we can compute the body forces using a simple matrix multiplication,

$$\begin{pmatrix} \mathbf{f} \\ \tau \end{pmatrix} = \begin{pmatrix} \mathbf{n}_1 & \mathbf{n}_2 & \cdots & \mathbf{n}_n \\ \mathbf{r}_1 \times \mathbf{n}_1 & \mathbf{r}_2 \times \mathbf{n}_2 & \cdots & \mathbf{r}_n \times \mathbf{n}_n \end{pmatrix} \begin{pmatrix} f_{N_1} \\ f_{N_2} \\ \vdots \\ f_{N_n} \end{pmatrix} \tag{3.3}$$

If we have multiple interacting bodies, a similar formula computes the forces on all of the bodies. For two interacting bodies, the forces are symmetric,

$$\begin{pmatrix} \mathbf{f}_1 \\ \tau_1 \\ \mathbf{f}_2 \\ \tau_2 \end{pmatrix} = \begin{pmatrix} \mathbf{n}_1 & \mathbf{n}_2 & \cdots & \mathbf{n}_n \\ \mathbf{r}_1 \times \mathbf{n}_1 & \mathbf{r}_2 \times \mathbf{n}_2 & \cdots & \mathbf{r}_n \times \mathbf{n}_n \\ -\mathbf{n}_1 & -\mathbf{n}_2 & \cdots & -\mathbf{n}_n \\ -\mathbf{r}_1 \times \mathbf{n}_1 & -\mathbf{r}_2 \times \mathbf{n}_2 & \cdots & -\mathbf{r}_n \times \mathbf{n}_n \end{pmatrix} \begin{pmatrix} f_{N_1} \\ f_{N_2} \\ \vdots \\ f_{N_n} \end{pmatrix} \tag{3.4}$$

which generalizes in the obvious way. To simplify notation, we will call the matrix $\mathbf{J}^T$, as it is the Jacobian of the constraint $\ddot{\mathbf{p}}_i = 0$. Likewise, we will use $\mathbf{f}$ to denote the vector $[f_{N_1}, f_{N_2}, \ldots]$. To convert this into linear ($\mathbf{a}_i$) and angular (α_i) accelerations, we need to construct the block diagonal mass matrix $\mathbf{M} = \text{diag}(\mathbf{M}_1, \mathbf{I}_1, \mathbf{M}_2, \mathbf{I}_2, \ldots)$, where $\mathbf{M}_i = m_i\mathbf{I}$ is a 3×3 diagonal mass matrix and $\mathbf{I}_i$ is the inertia tensor, which is stored in body local coordinates (and often diagonalized) and must be transformed into global coordinates at each timestep. Computing accelerations is simple,

$$\begin{pmatrix} \mathbf{a}_1 \\ \alpha_1 \\ \mathbf{a}_2 \\ \alpha_2 \\ \vdots \end{pmatrix} = \mathbf{M}^{-1}(\mathbf{J}^T\mathbf{f} + \mathbf{f}_0) \tag{3.5}$$

Here, $\mathbf{f}_0$ is a general term containing forces external to collision handling such as gravity, active control, and user forces. If we assume that we are using Euler's method to timestep dynamics, we can convert this into velocities at the end of the timestep,

$$\begin{pmatrix} \mathbf{v}_1^{t+\Delta t} \\ \omega_1^{t+\Delta t} \\ \mathbf{v}_2^{t+\Delta t} \\ \mathbf{v}_2^{t+\Delta t} \\ \vdots \end{pmatrix} = \begin{pmatrix} \mathbf{v}_1^t \\ \omega_1^t \\ \mathbf{v}_2^t \\ \mathbf{v}_2^t \\ \vdots \end{pmatrix} + \Delta t\mathbf{M}^{-1}(\mathbf{J}^T\mathbf{f} + \mathbf{f}_0) \tag{3.6}$$

What we are really concerned with, however, is the normal velocities at the contact points at the end of the timestep. We can convert from body velocities to contact point velocities using the formula

$$v_{N_i} = \mathbf{n}_i \cdot (\mathbf{v}_i + \alpha_i \times \mathbf{r}_i) \tag{3.7}$$

We can distribute the dot product,

$$v_{N_i} = \mathbf{n}_i \cdot \mathbf{a}_i + \mathbf{n}_i \cdot (\omega_i \times \mathbf{r}_i) \tag{3.8}$$

A classic identity for the triple product $\square \cdot (\square \times \square)$ is

$$\begin{aligned} \mathbf{n}_i \cdot (\omega_i \times \mathbf{r}_i) &= \det \begin{pmatrix} \mathbf{n}_i & \omega_i & \mathbf{r}_i \end{pmatrix} && (3.9) \\ &= \det \begin{pmatrix} \omega_i & \mathbf{r}_i & \mathbf{n}_i \end{pmatrix} && (3.10) \\ &= \omega_i \cdot (\mathbf{r}_i \times \mathbf{n}_i) && (3.11) \end{aligned}$$

For a single body, we can therefore compute the normal acceleration at each contact point using the formula,

$$\begin{pmatrix} a_{N_1} \\ a_{N_2} \\ \vdots \\ a_{N_n} \end{pmatrix} = \begin{pmatrix} \mathbf{n}_1^T & (\mathbf{r}_1 \times \mathbf{n}_1)^T \\ \mathbf{n}_2^T & (\mathbf{r}_2 \times \mathbf{n}_2)^T \\ \vdots & \vdots \\ \mathbf{n}_n^T & (\mathbf{r}_n \times \mathbf{n}_n)^T \end{pmatrix} \begin{pmatrix} \mathbf{v}_1 \\ \omega_1 \end{pmatrix} \tag{3.12}$$

which can be easily generalized to multiple bodies. Note that this is exactly the transpose of the matrix $\mathbf{J}^T$ defined in (3.4).

We now have a formula for converting all the way from normal forces applied at contact points to the post-timestep velocity $v_{N_i}^i t + \Delta t$ at those points,

$$\begin{pmatrix} v_{N_1}^{t+\Delta t} \\ v_{N_2}^{t+\Delta t} \\ \vdots \\ v_{N_n}^{t+\Delta t} \end{pmatrix} = \mathbf{J} \begin{pmatrix} \mathbf{v}_1^t \\ \omega_1^t \\ \mathbf{v}_2^t \\ \mathbf{v}_2^t \\ \vdots \end{pmatrix} + \Delta t \mathbf{J} \mathbf{M}^{-1} \mathbf{J}^T \mathbf{f} + \Delta t \mathbf{J} \mathbf{M}^{-1} \mathbf{f}_0 \tag{3.13}$$

Thus, we can write relationship between the set of normal forces f_{N_i} applied to a pair of colliding bodies and the resulting post-impulse velocities $v_{N_i}^{t+\Delta t}$ as a linear relationship,

$$\mathbf{v}^{t+\Delta t} = \mathbf{A}\mathbf{f} + \mathbf{b} \tag{3.14}$$

where $\mathbf{A}$ is and $n \times n$ symmetric matrix, where n is the number of constraints. This same formula will work for any force which is applied at a point; in particular, by using a linear approximation to the friction cone we can include friction forces as well (see §3.6).

3.3 Time-stepping

Stewart and Trinkle [ST96] noted that formulating our contact force conditions in terms of impulses $f_{N_i} \Delta t$ and velocities instead of forces and accelerations avoids some situations where no valid contact forces exist under the Coulomb friction model. Specifically, under the standard Coulomb model dynamic friction is constrained to be proportional to the normal force. It is thus possible to construct a situation such that increasing friction means that a larger normal force is required to prevent interpenetration. However, if we increase the normal force, the Coulomb

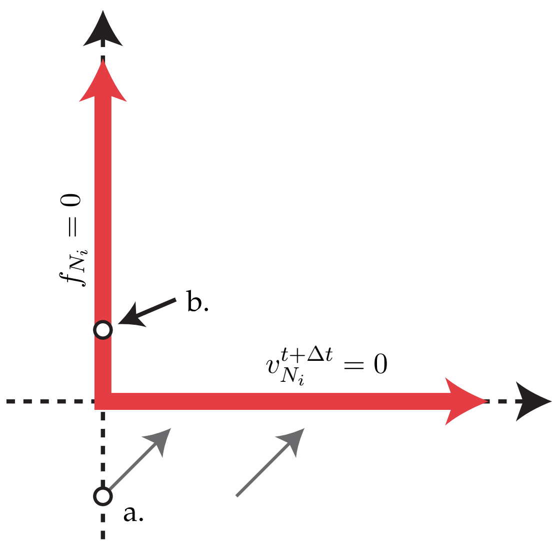


Figure 3.2: Linear complementarity problem for the normal force: ***(Right)*** *Constraints on the acceleration and force at a contact point i correspond to values of v_{N_i} and f_{N_i} which lie on the thick red line. When solving the linear complementarity problem, we begin with $f_{N_i} = 0$; if $v_{N_i} \geq 0$ (e.g., point a), we're done, but if $v_{N_i} < 0$ (point* **b***) then we need to increase f_{N_i} until we have moved v_{N_i} into the valid range.*

model stipulates that friction must increase as well, creating a positive feedback loop and ensuring that no valid solution exists; this is the classic Painlevé problem and discussions can be found in various places [Bar91, Ste00]. Posing the problem in terms of velocities and impulses stops the feedback loop: because friction impulses are required to oppose tangential velocity, they must cease growing once that velocity has been driven to zero.

Under Stewart's and Trinkle's timestepping scheme, object velocities $\{\mathbf{v}_i^{t+\Delta t}, \omega_i^{t+\Delta t}\}$ are computed at the next timestep. These are then used to compute the positions $\{\mathbf{x}_i^{t+\Delta t}, \Theta_i^{t+\Delta t}\}$:

$$\mathbf{v}_i^{t+\Delta t} = \mathbf{v}_i^t + \Delta t \mathbf{a}_i \tag{3.15}$$
$$\omega_i^{t+\Delta t} = \omega_i^t + \Delta t \alpha_i \tag{3.16}$$
$$\mathbf{x}_i^{t+\Delta t} = \mathbf{x}_i^t + \Delta t \mathbf{v}_i^{t+\Delta t} \tag{3.17}$$
$$\Theta_i^{t+\Delta t} = e^{[\Delta t \omega_i^{t+\Delta t}]}\Theta_i^t \tag{3.18}$$

When implementing this, linear and angular momenta should be stored internally in lieu of velocities as this saves us the trouble of accounting for precession; see Baraff's course notes for more details [Bar01].

3.4 LCP formulation for frictionless contact

Impulses $f_{N_i}\Delta t$ and velocities $v_{N_i}^{t+\Delta t} = v_{N_i}^t + \Delta t f_{N_i}$ must satisfy three conditions, namely:

Conditions for the normal force:

(a) bodies must not interpenetrate at the given contact point, that is, $v_{N_i}^{t+\Delta t} \geq 0$;
(b) contact forces must only repel and never attract, that is, $f_{N_i} \geq 0$; and
(c) contact forces can only occur when $v_{N_i}^{t+\Delta t} = 0$, that is, $(f_{N_i})(v_{N_i}^{t+\Delta t}) = 0$.

This last condition corresponds to the intuition that contact forces should not add mechanical energy to the system (e.g., we would not expect a block sitting on a table to suddenly jump up into the air). As $v_{N_i}^{t+\Delta t} \geq 0$, setting $f_{N_i} = 0$ when $v_{N_i}^{t+\Delta t} > 0$ ensures that each body does not do positive work on the other.

Note that in their original paper, Stewart and Trinkle posed (a) in terms of the separation between the bodies, requiring that the velocities be such that at the end of the timestep the bodies are not interpenetrating. We prefer to use velocities (as did Anitescu and Potra [AP96]); as we will see during Chapter 5, forcibly separating the bodies during backward simulation imparts a nonzero velocity that tends to grow without bound due to collision events. If we wanted to pose the problem in terms of positions (to reduce interpenetration) we note that this scheme can be emulated (and is in ODE) by changing $v_{N_i}^{t+\Delta t} \geq 0$ to $v_{N_i}^{t+\Delta t} \geq C$ for some constant C.

These three conditions combined with the linear relationship (3.14) form a linear complementarity problem, which can be written

$$0 \leq v_{N_i}^{t+\Delta t} \qquad \perp \qquad f_{N_j} \geq 0 \tag{3.19}$$

where the $\perp$ symbol is used to indicate the complementarity condition that only one of the two terms is allowed to be nonzero (equivalently, $v_{N_i}^{t+\Delta t} f_{N_j} = 0$).

3.5 Restitution

The model described in the previous section handles resting contact but cannot produce rebounds at impacts. Models for handling these impacts date back back to Newton, who stated his law of impact in terms of α, the *coefficient of restitution*, the ratio of pre- and post-collision velocities, which ranges from 0 to 1. A competing approach due to Poisson breaks collisions into two phases: first, during the compression phase, kinetic energy is stored as elastic deformation, and then during the restitution phase some fraction of that stored energy is turned back into motion. Under this model, during the first phase we record the impulse needed to halt the object and scale it by α to determine the impulse applied during the second phase [Ste00]. We will focus on the Newton model as it is simple to pose within the LCP formulation and is used within ODE. We note, however, that the Newton model is known to have certain problems that have been well-studied [Str90]. In particular, real-world restitution coefficients depend heavily on geometry in a manner that suggests that the storage of collision energy in elastic vibrations is an important factor in restitution [SH96].

Technically, only collisions with $\alpha = 1$ can be described as elastic, but we will use the term "elastic" to refer broadly to all collisions that are not completely inelastic, $0 < \alpha \leq 1$. This follows the terminology of Guendelman and colleagues [GBF03], but would be referred to as "partially elastic" in the physics literature.

To add elastic collisions to the LCP formulation in (3.19) requires only the simple modification [Ste98], (3.19),

$$0 \leq v_{N_i}^{t+\Delta t} + \alpha v_{N_i}^{t} \qquad \perp \qquad f_{N_i} \geq 0 \tag{3.20}$$

That is, we limit the allowed range of $v_{N_i}^{t+\Delta t}$ so it must be at least large enough to produce the desired elastic collision response. As before, the $\perp$ symbol indicates a complementarity condition,

$$(v_{N_i}^{t+\Delta t} + \alpha v_{N_i}^{t})(f_{N_i}) = 0 \tag{3.21}$$

Or, equivalently,

$$v_{N_i}^{t+\Delta t} \geq -\alpha v_{N_i}^{t} \text{ and } f_{N_i} = 0 \qquad \text{or} \qquad v_{N_i}^{t+\Delta t} = -\alpha v_{N_i}^{t} \text{ and } f_{N_i} \geq 0 \tag{3.22}$$

The decision of whether to apply an elastic or inelastic collision response is commonly made by comparing $v_{N_i}^{t}$ with a cutoff velocity ϵ; an elastic response only occurs if $-v_{N_i}^{t} > \epsilon$. This ensures that computation time is not spent applying barely perceptible elastic micro-collisions to bodies. This will be important for time-reversed simulation because a means for objects to transition from moving to static will be necessary for objects to transition the other direction when simulation backwards (and it is likely that "sitting on the floor" will be a common initial condition for artists to specify).

3.6 Frictional LCP

For friction, we associate with each contact point i a pair of vectors $\{\mathbf{u}, \mathbf{w}\}$ defining a pyramidal approximation to the friction cone (in Figure 3.1, left we can see one of these vectors at each contact point). Note that more vectors could be used to provide a better approximation at the cost of increased computational cost (see §3.6.1 for more discussion of this). Associated with the friction constraints are forces $f_{\mathbf{w}_i}\mathbf{w}_i$ and $f_{\mathbf{u}_i}\mathbf{u}_i$ which act at contact points, similar to the way $f_{N_i}\mathbf{n}_i$ acts to prevent interpenetration. The constraints are derived from the standard Coulomb model:

Conditions for the friction force:

(a) friction must always oppose velocity; that is, if $v_{\mathbf{w}_i}^{t+\Delta t} \leq 0$ then $f_{\mathbf{w}_i} \geq 0$ and vice versa,

(b) friction forces are bounded by a constant fraction μ of the normal force, $|f_{\mathbf{w}_i}| + |f_{\mathbf{u}_i}| \leq \mu f_{N_i}$, and

(c) friction must be maximal if the resulting tangential velocity is nonzero,

$$(v_{\mathbf{w}_i}^{t+\Delta t}) \perp (\mu f_{N_i} - |f_{\mathbf{w}_i}|)$$

This last condition is related to the *principle of maximal dissipation*, which is normally stated in terms of velocities, namely that the post-timestep velocity should be the one that is minimal subject to the Coulomb constraints [Ste00]. For our purposes, though, it is easier to state it in terms of forces; intuitively, the frictional force should maximally oppose the tangential velocity of the object [Bar91].

The complementarity conditions for friction can be stated as:

$$0 \leq \mu f_{N_i} - (|f_{\mathbf{w}_i}| + |f_{\mathbf{u}_i}|) \qquad \perp \qquad v_{\mathbf{w}_i}^{t+\Delta t} \tag{3.23}$$

along with the following condition, derived from (a) above:

$$f_{N_i} v_{\mathbf{w}_i}^{t+\Delta t} \leq 0 \tag{3.24}$$

The relationship between force and velocity for a single friction direction can be seen visually in Figure 3.3, left, where valid values lie on the red line. Note that we can reduce the diagram in Figure 3.3 to the one in Figure 3.2 via a simple modification of the bounds on $f_{\mathbf{w}_i}$; see Figure 3.3, right. Thus, both constraints can be handled within the same framework [Bar94, Smi06], although the fact that the bounds on friction are proportional to normal force rather than constant require some extra care.

3.6.1 ODE's friction model

ODE uses two approximations to accelerate the computation of friction forces. The first is to decouple the two directions in the friction cone; that is, instead of requiring that

$$|f_{\mathbf{u}_i}| + |f_{\mathbf{w}_i}| \leq \mu f_{N_i} \tag{3.25}$$

as in the standard polyhedral model of friction, ODE requires only that the individual friction directions be bounded,

$$|f_{\mathbf{u}_i}| \leq \mu f_{N_i} \qquad |f_{\mathbf{w}_i}| \leq \mu f_{N_i} \tag{3.26}$$

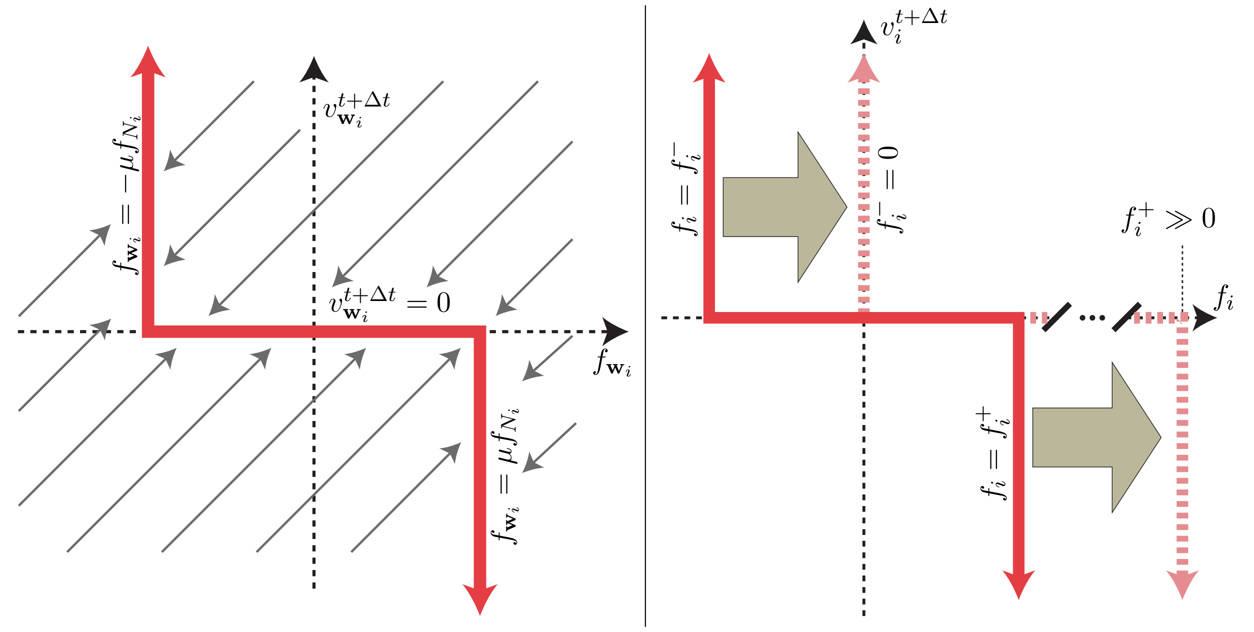

$v_{\mathbf{w}_i}^{t+\Delta t}$
$f_{\mathbf{w}_i} = -\mu f_{N_i}$
$v_{\mathbf{w}_i}^{t+\Delta t} = 0$
$f_{\mathbf{w}_i}$
$f_{\mathbf{w}_i} = \mu f_{N_i}$
$v_i^{t+\Delta t}$
$f_i = f_i^-$
$f_i^- = 0$
$f_i^+ \gg 0$
f_i
$f_i = f_i^+$

Figure 3.3: Frictional linear complementarity problem: Left: *We simplify by showing only a single friction direction; here, the friction force $f_{\mathbf{w}_i}$ is required to lie on the red line. Intuitively, friction should either oppose velocity or halt tangential motion altogether; the lines $f_{\mathbf{w}_i} = \pm\mu f_{N_i}$ correspond to the first condition while the segment $v_{\mathbf{w}_i}^{t+\Delta t} = 0$ corresponds to the second. The goal of the LCP solver is to modify $f_{\mathbf{w}_i}$ until the point $(f_{\mathbf{w}_i}, v_{\mathbf{w}_i}^{t+\Delta t})$ lies on the red line; this corresponds to moving along the diagonal lines shown here.* (Credit: diagram inspired by one in [Smi06].) **Right:** *If we replace $-\mu f_{N_i} \leq f_{\mathbf{w}_i} \leq \mu f_{N_i}$ with generic force bounds $f_i^- \leq f_i \leq f_i^+$, we can imagine converting the frictional LCP into the normal force LCP in Figure 3.2 by allowing $f_i^- \to 0$ and $f_i^+ \to \infty$.*

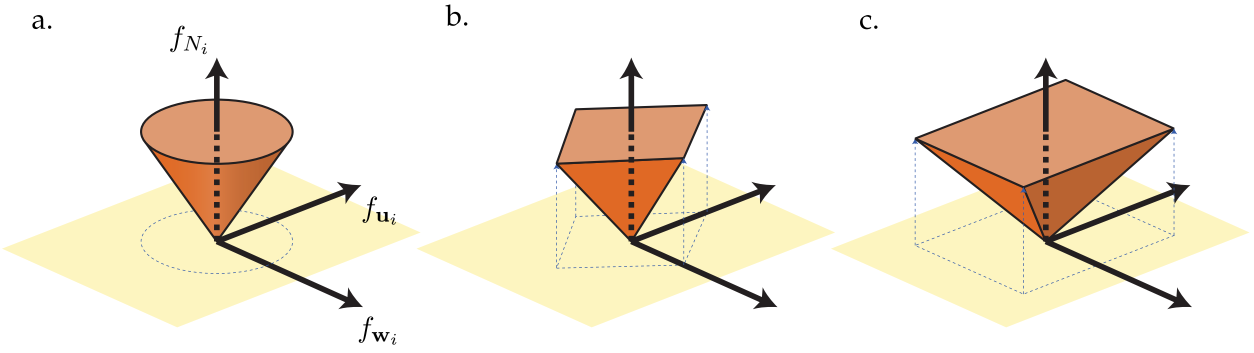


Figure 3.4: Approximations to the friction cone: (a) *Under the Coulomb model, the magnitude of the friction force is bounded above by a fraction of the normal force, $(f_{\mathbf{u}_i}^2 + f_{\mathbf{w}_i}^2)^{1/2} \leq \mu f_{N_i}$.* **(b)** *The polyhedral approximation used by Stewart and Trinkle and others restricts $|f_{\mathbf{u}_i}| + |f_{\mathbf{u}_i}| \leq \mu f_{N_i}$.* **(c)** *ODE [Smi06] decouples the two coefficients, $|f_{\mathbf{u}_i}| \leq \mu f_{N_i}$ and $|f_{\mathbf{w}_i}| \leq \mu f_{N_i}$. This simplifies implementation at the risk of allowing overly large friction impulses; unlike the polyhedral approximation, this method cannot be made to converge to the true friction cone by adding additional directions.*

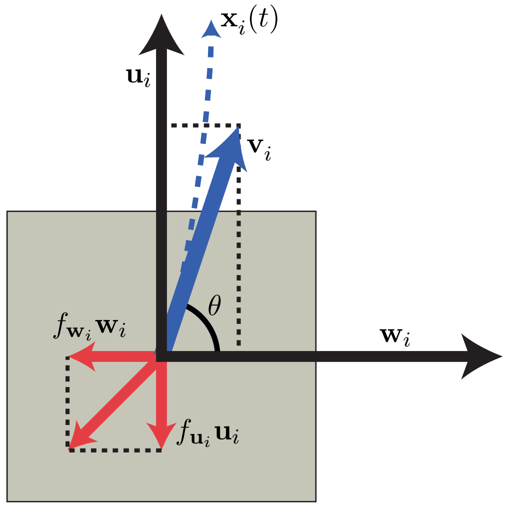


Figure 3.5: Limitation of the decoupled friction cone approximation: *By decoupling the two friction directions $\mathbf{u}$ and $\mathbf{w}$, we can generate spurious accelerations. Here, we see an overhead view of a box sliding in the plane with velocity $\mathbf{v}_i$. If we apply the maximum friction impulse μf_{N_i} along each of the friction directions $\mathbf{u}$ and $\mathbf{w}$, the summed impulse fails to correctly oppose $\mathbf{v}_i$; the result is that the box's path will curve as it slows to a stop.*

See also Figure 3.4. Decoupling these forces simplifies implementation somewhat. One drawback of this method is that while we can make the polyhedral approximation (3.25) approach the true friction cone by adding additional friction directions, additional directions do not cause (3.26) to converge to anything (in fact, the total friction can grow without bound). The second drawback can be seen in figure Figure 3.5: because the object's tangent velocity is projected independently on each of the two axes, spurious direction changes can result (see Figure 3.5). We can eliminate the worst effects (for both forward and backward simulation) by using the relative velocity at $\mathbf{p}_i$ as one of the two friction directions.

The second approximation ODE uses relates to the relationship between friction bounds and the normal force. Making these proportional during the LCP solve has two major consequences. First, the major operation that the LCP solver takes that can be seen in the grey lines in both Figure 3.2 and Figure 3.3 is to take a step from the invalid region into the valid region by changing f_i. For friction, the vertical lines in Figure 3.3 are functions of other forces in the system, all of which may be changing to accommodate the increased friction. This complicates the logic somewhat, but from an implementation perspective the most important effect is that the matrices solved are no longer symmetric. ODE gets significant mileage out of a fast update to the factorization $\mathbf{A} = \mathbf{LDL}^T$

based on the recursive definition,

$$\begin{pmatrix} \mathbf{A}_{n-1} & \mathbf{a}_n \\ \mathbf{a}_n^T & a_{n,n} \end{pmatrix} = \begin{pmatrix} \mathbf{L}_{n-1} & \mathbf{0} \\ \mathbf{l}_n^T & \ell_{n,n} \end{pmatrix} \begin{pmatrix} \mathbf{D}_{n-1} & \mathbf{0} \\ \mathbf{0} & d_{n,n} \end{pmatrix} \begin{pmatrix} \mathbf{L}_{n-1}^T & \mathbf{l}_n \\ \mathbf{0} & \ell_{n,n} \end{pmatrix} \tag{3.27}$$

$$\mathbf{L}_{n-1}\mathbf{D}_{n-1}\mathbf{L}_{n-1}^T = \mathbf{A}_{n-1} \tag{3.28}$$

$$\mathbf{L}_{n-1}\mathbf{D}_{n-1}\mathbf{l}_n = \mathbf{a}_n^T \tag{3.29}$$

$$\mathbf{l}_n^T\mathbf{D}_{n-1}\mathbf{l}_n + d_{n,n}\ell_{n,n}^2 = a_{n,n} \tag{3.30}$$

Here, the $\mathbf{L}_{n-1}\mathbf{D}_{n-1}\mathbf{L}_{n-1}^T$ in (3.28) is computed recursively and solving for $\mathbf{l}_n$ in (3.29) requires only back-substitution. Davis calls the equivalent formulation of the Cholesky factorization an *up-looking Cholesky factorization* [Dav06], although this terminology does not seem to be standard. This update is particularly simple to code because it does not require pivoting, a classic bugbear for methods that update matrix factorizations (although it is worth noting that *removing* a row/column pair requires more care). However, the $\mathbf{LDL}^T$ only makes sense for symmetric matrices, and for reasons that will become clear in §3.7, matrices solved in the LCP solver are only symmetric if the bounds on f_i are not dependent on other f_j (where $j \neq i$). To get around this restriction, ODE breaks the LCP solve into two phases. First, normal forces are computed as if all contacts are frictionless. These normal forces are used to compute friction bounds μf_{N_i} for each contact. In the second, both friction and normal forces are computed, but friction bounds are held fixed.

Unfortunately, this approximate friction model produced spurious torques when applied in backward simulation to cases as simple as a block sliding across a surface, and as a result we found it necessary to modify the LCP solver so that the normal and friction forces were tightly coupled and computed simultaneously. We will return to this with a specific example after discussing how we handle friction for backward simulation.

3.7 Solving the LCP Problem

The goal of the LCP solver is to adjust each of the $\{f_{\{N_i,\mathbf{u}_i,\mathbf{w}_i\}}, v_{\{N_i,\mathbf{u}_i,\mathbf{w}_i\}}^{t+\Delta t}\}$ pairs until each lies on the red line shown in Figure 3.3. For the purposes of this discussion, we will simplify notation somewhat. As the LCP solver handles friction and normal constraints mostly the same (see Figure 3.3, right), we will re-index the various $f_{\{N_i,\mathbf{u}_i,\mathbf{w}_i\}}$ as f_i and similarly the various $v_{\{N_i,\mathbf{u}_i,\mathbf{w}_i\}}^{t+\Delta t}$ will become v_i. We label the bounds on f_i using the $\{f_i^-, f_i^+\}$ notation that is used in Figure 3.3, right – but it is important to remember that the bounds can be functions of other forces in the system, that is, $f_i^+ = \mu_i f_j$ for $j \neq i$. The resulting LCP problem becomes

$$A_{ij}f_j + b_i = v_i \qquad \text{or} \qquad \mathbf{Af} + \mathbf{b} = \mathbf{v} \tag{3.31}$$

along with the LCP condition,

$$(f_i - f_i^-) \perp v_i \qquad \text{or} \qquad (f_i - f_i^+) \perp v_i \tag{3.32}$$

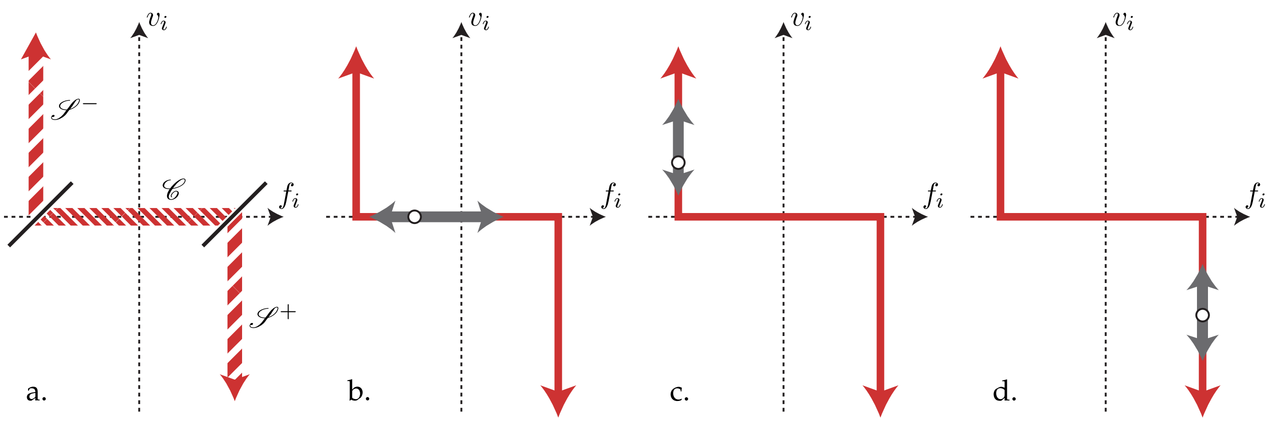


Figure 3.6: Solving the LCP problem: **(a)** *In the ODE LCP solver, the variable i is placed in one of 3 sets based on the values f_i and v_i.* **(b)** *For variables $i \in \mathscr{C}$ we vary f_i to maintain the constraint $v_i = 0$, but if f_i reaches f_i^+ we move i into $\mathscr{S}^+$ before continuing; a similar process is used if f_i reaches f_i^-.* **(c-d)** *For variables $i \in \mathscr{S}^+ \cup \mathscr{S}^-$ we maintain $f_i = f_i^+$ or $f_i = f_i^-$ as appropriate, but move i into $\mathscr{C}$ if v_i reaches 0.*

And (for friction) the additional condition

$$f_i v_i \leq 0 \tag{3.33}$$

Note that this last condition is safe to include even for normal forces, as the combination of $f_i^- = 0$ with the LCP condition $(f_i - f_i^-) \perp v_i$ imply that $f_i v_i = 0$ trivially.

The LCP constraints described up to now can be all be represented in this framework. Recall the formula for transforming forces into post-timestep velocities,

$$\mathbf{v}^{t+\Delta t} = \mathbf{v}^t + \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{f} + \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_0 \tag{3.34}$$

For both the normal conditions (3.19) and the friction conditions (3.23), the LCP condition on the post-timestep velocity is that $v_i^{t+\Delta t} = 0$ if $f_i \neq f_i^-$ and $f_i \neq f_i^+$. Thus, these can be transformed into the form of (3.31) by setting

$$\mathbf{A} = \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T \qquad\qquad \mathbf{b} = \mathbf{v}^t + \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_0 \tag{3.35}$$

For the restitution constraint (3.20), this will not work because the LCP condition is of the form

$$f_i^- \leq f_i \qquad\qquad \perp \qquad\qquad v_i - v_i^{goal} \tag{3.36}$$

Here the notation v_i^{goal} indicates that the LCP solver is to drive the post-timestep velocity toward the given goal value. We can convert (3.36) into (3.32) by moving v_i^{goal} into $\mathbf{b}$,

$$\mathbf{b} = \mathbf{v}^t + \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_0 - \mathbf{v}^{goal} \tag{3.37}$$

The algorithm we use to solve (3.31)-(3.33) is known as *Lemke's algorithm* [CPS92], and our implementation is based on code in ODE.

Lemke's algorithm handles the constraints one at a time: first, it takes the pair $\{f_1, v_1\}$ and moves them into the valid region, then it handles $\{f_2, v_2\}$, and so forth. While modifying the variables $\{f_i, v_i\}$, it ensures that the pairs $\{f_j, v_j\}$ for $j < i$ do not leave the valid region. Because $\mathbf{A}$ is positive semi-definite, v_i increases with f_i; we therefore move toward the valid region along the lines shown in gray in Figure 3.3. It is the step of ensuring that the other variables $\{f_j, v_j\}$ for $j < i$ that complicates the computation. To discuss this, we need some extra notation; looking again at Figure 3.3, we notice that the red line signifying the valid region can be broken into three parts: two vertical lines and one horizontal line. We can classify constraints i based on which region they fall in: variables that lie on the horizontal line are placed in set $\mathscr{C}$, for *constrained*, so-called because v_i lies on the constraint manifold. Variables on either vertical line belong to the set $\mathscr{S}$, for *slack*, a term borrowed from linear programming that refers to the slack in v_i. We can further break $\mathscr{S}$ down into $\mathscr{S}^+$ and $\mathscr{S}^-$ as shown in Figure 3.6.

The challenging part of Lemke's algorithm is ensuring that as we drive variable i toward the valid region, each of the variables that we have already handled remains in its own valid region. How we handle each variable $j < i$ depends on its classification. Variables j in $\mathscr{S}$ are relatively simple, as they can be allowed to vary along the vertical bars seen in Figure 3.6, (c-d). Thus, we can simply hold f_j fixed at its original value, and need worry only making sure that j gets transferred into $\mathscr{C}$ if v_j reaches 0 (this will get slightly more complicated in a moment, but this description is sufficient for now). Variables in $\mathscr{C}$, however, require more care; adding force along f_i will cause any number of v_j in the system to change, and we cannot allow any $j \in \mathscr{C}$ to slide off the line $v_j = 0$. Fortunately, we know exactly how changing f_i will affect v_j for $j \neq i$, as this information is encoded in the system matrix $\mathbf{A}$; if we increase f_i by Δf_i, v_j will increase by $\alpha \mathbf{A}_{j,i}$. The basic idea is that we can compute a set of Δf_j for $j < i$ that when added to f_j will act to counter any changes in v_j, thus keeping each v_j on the line $v_j = 0$ (Figure 3.6, b). To make this concrete, we want to find the Δf_j such that

$$\sum_{k \in \mathscr{C}} A_{jk} \Delta f_k + A_{ij} \Delta f_i = 0 \tag{3.38}$$

We can solve this for any arbitrary Δf_i by setting Δf_i to 1 and scaling the solution by Δf_i after the fact. Thus, we need to solve the following linear system,

$$\sum_{k \in \mathscr{C}} A_{jk} \Delta f_k = -A_{ij}, \qquad j \in \mathscr{C} \tag{3.39}$$

or, equivalently,

$$\mathbf{A}_{\mathscr{C},\mathscr{C}} \Delta \mathbf{f}_{\mathscr{C}} = -\mathbf{A}_{i,\mathscr{C}} \tag{3.40}$$

Because $\mathbf{A}$ is symmetric, this system can be solved quickly using the LDL^T factorization as described in §3.6.1. However, we have glossed over one important detail: recall that to accommodate friction we allowed f_j^+ to be a linear function of some f_j for $j \neq k$, that is, $f_j = \mu_j f_k$ (likewise, $f_j^- = -\mu_j f_k$). This becomes an issue if $j \in \mathscr{S}$; to keep f_j on the vertical line $f_j^+ = \pm\alpha_j f_k$ then we must modify f_j whenever we change f_k. Specifically, if $j \in \mathscr{S}^+$, then when we add Δf_k to f_k we must simultaneously add $\alpha_j \Delta f_k$ to f_j to preserve the constraint $f_j = \alpha f_k$. If $j \in \mathscr{S}^-$ instead, the sign is inverted, as increasing f_k means we must *decrease* f_j.

We need to adjust (3.40) to account for the dependence between f_k and f_j. We introduce the notation $\mathscr{N}(j) = k$ which we will use to indicate that $f_j^+ = \mu_j f_{\mathscr{N}(k)}$ and $f_i^- = -\mu_j f_{\mathscr{N}(j)}$. Recalling that $\mathbf{A}$ mediates between $\mathbf{f}$ and $\mathbf{v}$, we can add an extra term to (3.40) that accounts for the dependence of f_j^+ on f_k,

$$\mathbf{A}_{\mathscr{C},\mathscr{C}}\, \Delta \mathbf{f}_{\mathscr{C}} + \sum_{\substack{j \in \mathscr{S} \\ \mathscr{N}(j) \in \mathscr{C}}} \mu \left(\operatorname{sgn} f_j\right) \Delta f_{\mathscr{N}(j)}\, \mathbf{A}_{\mathscr{C},j} = -\mathbf{A}_{i,\mathscr{C}} \tag{3.41}$$

The notation makes (3.41) look more complicated than it actually is; in practice, we need only to walk through the variables in the system and add the appropriately scaled $\mathbf{A}_{\mathscr{C},j}$ to the kth column of $\mathbf{A}_{\mathscr{C}}$ if $\mathscr{N}(j) = k$ and $j \in \mathscr{S}$. From the implementer's point of view, this is important because it breaks the symmetry of $\mathbf{A}$ and adds a new type of update that must be performed as variables are moved in and out of $\mathscr{S}$.

3.7.1 Fast QR updates

In the solver loop, we must repeatedly solve an equation of the form $\mathbf{Ax} = \mathbf{b}$, making only small changes to $\mathbf{A}$ each time. Specifically, we perform three kinds of updates:

1. *Adding a row and column.* Through judicious choice of indices, we can easily ensure that the added row and column always appear at the end of the matrix; that is, we make the following change,

$$\begin{pmatrix}\mathbf{A}_{1:n,\,1:n}\end{pmatrix} \rightarrow \begin{pmatrix}\mathbf{A}_{1:n,\,1:n} & \mathbf{v}_{1:n} \\ \mathbf{u}_{1:n}^T & w\end{pmatrix} \tag{3.42}$$

2. *Removing a row and column.* In this case, the row/column may be anywhere in the matrix; that is, we make the transformation

$$\begin{pmatrix}\mathbf{A}_{1:j-1,\,1:j-1} & \mathbf{A}_{1:j-1,\,j} & \mathbf{A}_{1:j-1,\,j+1:n} \\ \mathbf{A}_{j,\,1:j-1} & \mathbf{A}_{j,\,j} & \mathbf{A}_{j,\,j+1:n} \\ \mathbf{A}_{j+1:n,\,1:j-1} & \mathbf{A}_{j+1:n,\,j} & \mathbf{A}_{j+1:n,\,j+1:n}\end{pmatrix} \rightarrow \begin{pmatrix}\mathbf{A}_{1:j-1,\,1:j-1} & \mathbf{A}_{1:j-1,\,j+1:n} \\ \mathbf{A}_{j+1:n,\,1:j-1} & \mathbf{A}_{j+1:n,\,j+1:n}\end{pmatrix} \tag{3.43}$$

3. *Changing a column.* In this case, we need to adjust a column by adding some δ to it; that is, we make the change,

$$\begin{pmatrix} | & | & | \\ \mathbf{A}_{1:n,\,1:j-1} & \mathbf{A}_{1:n,\,j} & \mathbf{A}_{1:n,\,j+1:n} \\ | & | & | \end{pmatrix} \rightarrow \begin{pmatrix} | & | & | \\ \mathbf{A}_{1:n,\,1:j-1} & \mathbf{A}_{1:n,\,j} + \delta_{1:n} & \mathbf{A}_{1:n,\,j+1:n} \\ | & | & | \end{pmatrix} \tag{3.44}$$

It would be preferable to exploit the structure of the problem to speed up convergence. ODE uses a fast update based on a recursive definition of the Cholesky factorization, but in our case the matrices are not symmetric. Baraff used a package for doing fast LU updates, but this package does not seem to be publicly available. Regardless, LU updates are not always numerically stable and notoriously difficult to implement correctly due to pivoting requirements [GL96, p. 606]. Instead, we use QR updates which have the same asymptotic complexity as LU updates but are more numerically stable and (by comparison) quite simple to implement.

As a review, the QR factorization of a matrix is $\mathbf{A} = \mathbf{QR}$ where $\mathbf{Q}$ is an orthonormal basis for the column space and $\mathbf{R}$ is upper triangular. We can solve the equation $\mathbf{Ax} = \mathbf{b}$ using

$$\begin{aligned} \mathbf{QRx} &= \mathbf{b} && (3.45) \\ \mathbf{Rx} &= \mathbf{Q}^T\mathbf{b} && (3.46) \end{aligned}$$

and then use back-substitution to solve for $\mathbf{x}$.

Golub and Van Loan establish the basics of QR updating; however, as their description is somewhat terse and does not deal with the specific cases seen in the LCP solver, we will describe in some detail how to perform the updates for the three cases noted above.

The primary tool used in updating matrices are *Givens rotations*. These rotations are transforms of the form

$$G(i,\theta) = \begin{pmatrix} 1 & & & & & & \\ & \ddots & & & & & \\ & & 1 & & & & \\ & & & \cos\theta & \sin\theta & & & \\ & & & -\sin\theta & \cos\theta & & & \\ & & & & & 1 & & \\ & & & & & & \ddots & \\ & & & & & & & 1 \end{pmatrix} \begin{array}{l} \\ \\ \\ \leftarrow \quad i \\ \leftarrow \quad i+1 \\ \\ \\ \\ \end{array} \tag{3.47}$$

where θ is chosen such that $G(\theta, i)^T$ zeroes out the $(i, i+1)$ matrix entry, that is,

$$\begin{pmatrix} \cos\theta & \sin\theta \\ -\sin\theta & \cos\theta \end{pmatrix}^T \begin{pmatrix} * & * & \dots & * & * & \dots & * & * \\ * & * & \dots & * & * & \dots & * & * \end{pmatrix} = \begin{pmatrix} * & * & \dots & * & * & \dots & * & * \\ * & * & \dots & 0 & * & \dots & * & * \end{pmatrix} \tag{3.48}$$

These rotations are orthonormal and so

$$\mathbf{QR} = \mathbf{QGG}^T\mathbf{R} = (\mathbf{QG})(\mathbf{G}^T\mathbf{R}) \tag{3.49}$$

which preserves the orthogonality of $\mathbf{Q}$. The other useful property of Givens rotations is that applying one to both $\mathbf{Q}$ (on the right) and $\mathbf{R}$ (on the left) costs only $O(12n)$ flops for $n \times n$ matrices.

Givens rotations have very good cache performance, and are performed extremely quickly for dense matrices by the BLAS function `drot`. However, as $\mathbf{R}$ is upper triangular, we store it using a packed format which leaves out all the below-diagonal entries [DCHH88]; this means that the stock `drot` cannot be used. It was useful in our case to take the time to implement a carefully optimized `drot` which works for packed matrices. Note that standard implementations of QR

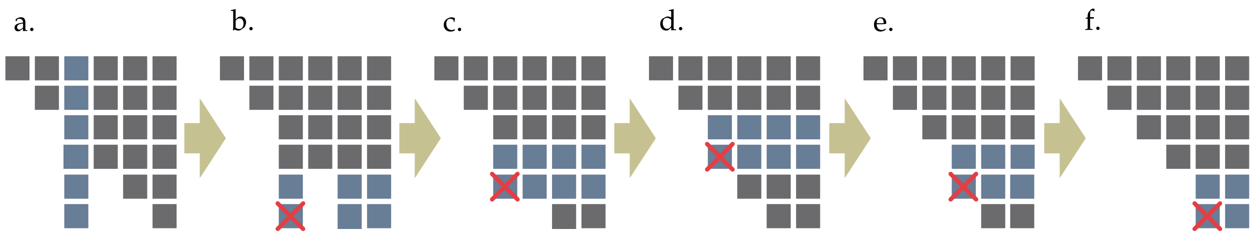


Figure 3.7: Using Givens rotations to update a single column: *First, a series of rotations* **(a-c)** *eliminates the entries in the "spike" column of* $\mathbf{R}$*, then a second series of rotations* **(e-f)** *zeroes out entries below the diagonal.*

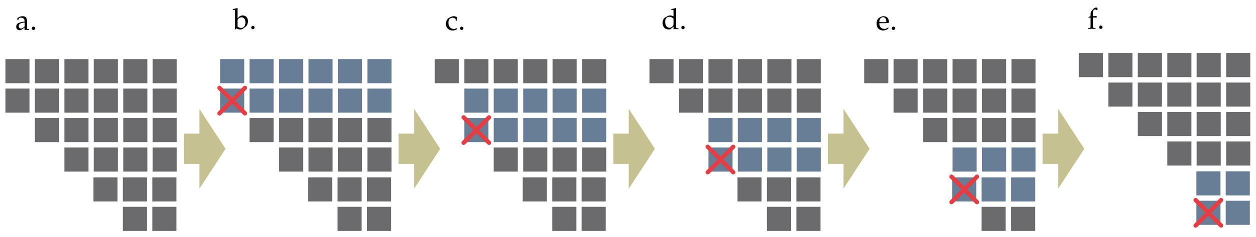


Figure 3.8: Applying Givens rotations when adding a row and column: *A series of rotations can be applied to* $\mathbf{R}$ *to eliminate entries below the diagonal.*

decomposition do not generally explicitly store the Q matrix but instead the minimal amount of information needed to reconstruct it. It is possible to use the same storage format when performing QR updates [LH87] but this is trickier to code and additional work would be necessary to combine this minimal storage with all the different kinds of updates required.

Updating a column: Each column of $\mathbf{A}$ can be represented as a linear combination of the columns of $\mathbf{Q}$,

$$\mathbf{A}_{1:n,\,i} = \sum_j \mathbf{Q}_{1:n,\,j} \mathbf{R}_{j,\,i} \tag{3.50}$$

It follows that to modify column i in $\mathbf{A}$, we need only modify the appropriate coefficients in $\mathbf{R}$ by adding $\mathbf{Q}^T\delta$ to column i. However, this will result in an $\mathbf{R}'$ matrix that is no longer lower triangular (it has a "spike" in the ith column; see Figure 3.7, a). Through a series of Givens rotations we first zero out all the entries in column i (Figure 3.7, a-c). However, this process will introduce nonzero entries below the main diagonal of $\mathbf{R}'$, so the next series of Givens rotations eliminates these as well (Figure 3.7, d-f). The number of Givens rotations required is at most $2n$, so the total update is $O(n^2)$. Note that we can perform this modification in place if use scratch arrays of length n to store column i and the entries that appear below the diagonal.

Adding a row and column: This is simple enough, as we observe that

$$\begin{pmatrix} \mathbf{A} & \mathbf{u} \\ \mathbf{v}^T & w \end{pmatrix} = \begin{pmatrix} \mathbf{0} & \mathbf{Q} \\ 1 & \mathbf{0} \end{pmatrix} \begin{pmatrix} \mathbf{v}^T & w \\ \mathbf{R} & \mathbf{Q}^T\mathbf{u} \end{pmatrix} \tag{3.51}$$

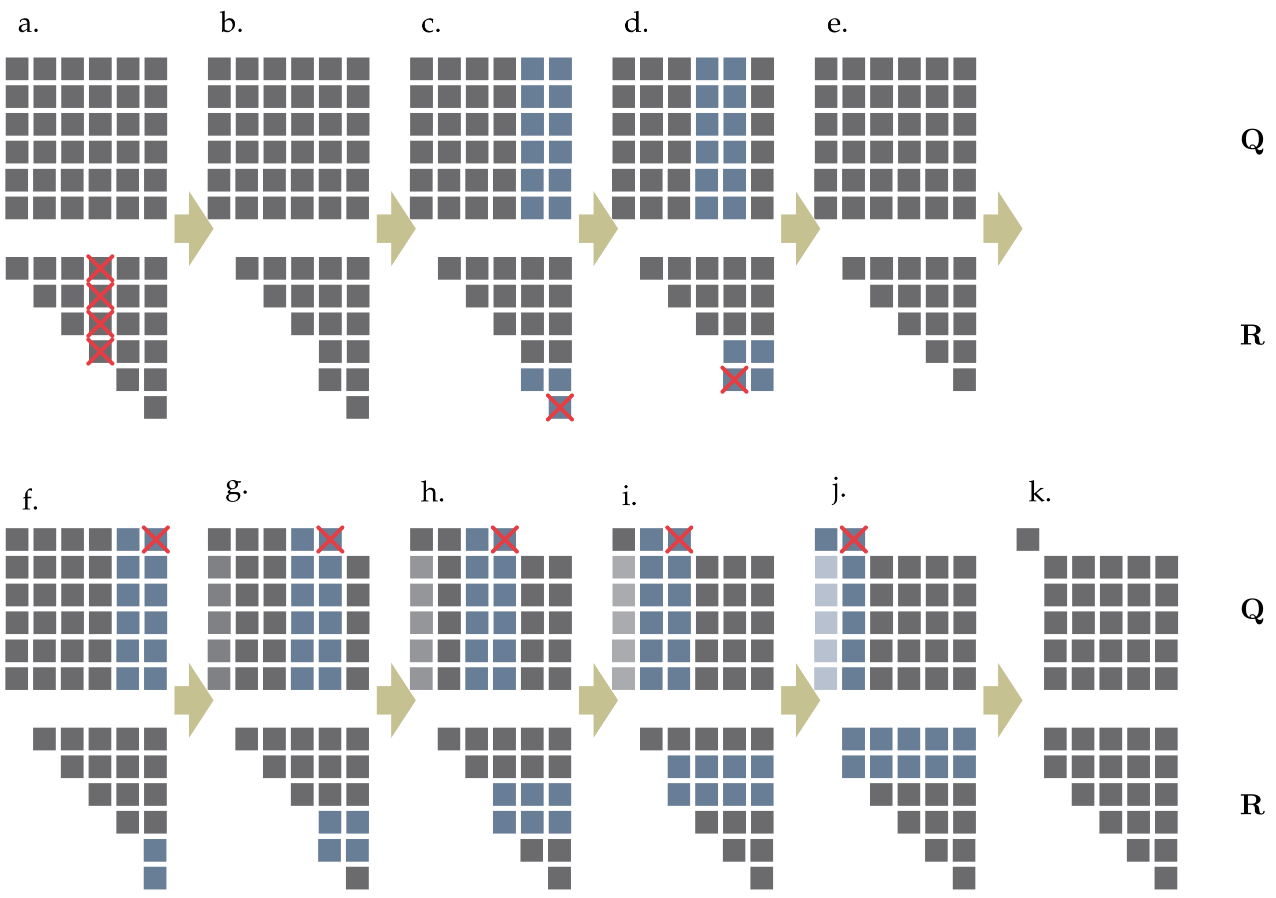


Figure 3.9: Applying Givens rotations when deleting a row and column: *After removing the column from* **R (a)** *we apply a series of Givens rotations* **(c-d)** *to make* **R** *upper triangular. However, the resulting* **Q (e)** *is too large; a series of rotations can be used to zero out all but the first entry of the first row of* **Q (f-j)**, *after which it is safe to remove the first row/column of* **Q** *and the first row of* **R (k)**.

This produces a matrix with entries below the main diagonal, which can be made upper triangular using Givens rotations as shown in Figure 3.8.

Removing a row and column: If we were only removing the column i from $\mathbf{A}$, we could simply drop the the corresponding column from $\mathbf{R}$ and perform a series of Givens rotations to make the matrix upper triangular (see Figure 3.9, a-e). At this point, we could eliminate the last column of $\mathbf{Q}$ and we would be done. To remove the row, however, we need to be more careful, because the obvious step of dropping a row from $\mathbf{Q}$ destroys orthogonality. Therefore, we first permute $\mathbf{Q}$ so that row i appears first in $\mathbf{Q}$ and then use Givens rotations to convert the first row of $\mathbf{Q}$ into $\begin{pmatrix}1 & 0 & 0 & \dots\end{pmatrix}$. As before, the rotations act on both $\mathbf{Q}$ and $\mathbf{R}$; however, in this case, we choose θ so that it zeroes out entries in $\mathbf{Q}$ instead of $\mathbf{R}$ (Figure 3.9, f-l). Note that when we are finished, the first column of $\mathbf{Q}$ also becomes $\begin{pmatrix}1 & 0 & 0 & \dots\end{pmatrix}$ as a result of orthonormality. At this point, we can freely drop both the first column and row of $\mathbf{Q}$ and the first row of $\mathbf{R}$ to get the desired result.

Chapter 4

Many-Worlds Browsing

The core of the Many-Worlds Browsing approach is the ability to compute and display many example motions to the user. This needs to be as fast as possible for the system to be really usable; forcing the user to wait several minutes before seeing any results has a substantial impact on productivity. To ensure fast turnarounds, we compute the examples in parallel on a cluster of machines (Figure 4.1). The sampling itself is trivial to parallelize, but the streaming requires that motions be compressed (§4.3) to ensure that throughput is not limited by the available network bandwidth.

4.1 Sampling plausible worlds

Before we can compute examples, the user must specify the objects in the scene and their initial state. Users can randomize the initial conditions of objects (or any other property in the system) using a simple interface. To allow scenes to be built up, the user is permitted to insert or remove objects even after much of the motion has been computed; we will discuss how this is handled in §4.5.

4.1.1 Plausibility

It is important that all the samples we generate be acceptable as answers to user queries, as it would be unreasonable to expect users of our system to sort through the figurative haystack of hundreds of poor-quality simulations to find their 'needle.' To ensure this, we rely on studies of plausibility performed by other researchers. Specifically, O'Sullivan and colleagues [ODGK03] examined people's ability to detect perturbations applied during impacts and found that people were unable to detect distortions of the post-collision linear velocity of up to 40% in magnitude and 20 degrees in angle (depending on various other factors). Similar tests for angular velocity found that distortions of up to 20% went undetected, depending somewhat on object shape.

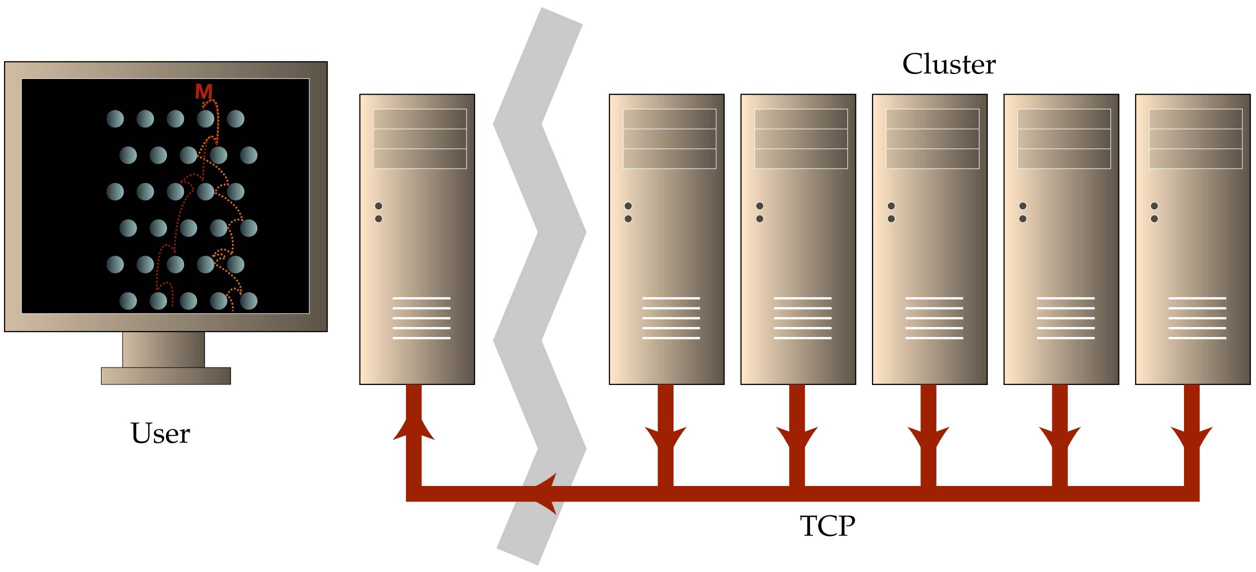


Figure 4.1: Streaming examples via TCP: *Sampling and refinement is performed in parallel on a cluster of machines (seen at right) and then streamed back to the user for display. As a result, minimizing the bandwidth consumed by each sample is key to minimizing the time users must wait to see usable motion.*

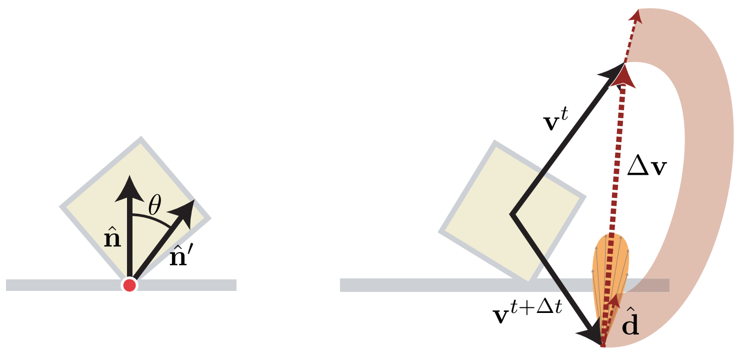


Figure 4.2: Applying perturbations at collision events. Left: *The application of small perturbations to collision normals was first introduced by Barzel and colleagues [BHW96] and then adopted by both Chenney and Forsyth [CF00] and Popović and colleagues [PSE$^+$00]. Here, the original collision normal* $\hat{\mathbf{n}}$ *is perturbed by an angle* θ *to produce the adjusted collision normal* $\hat{\mathbf{n}}'$. **Right:** *In our scheme, we modify post-collision velocities directly rather than indirectly through collision normals. We apply a perturbation impulse* $m\hat{\mathbf{d}}$ *to the post-impact velocity* $\mathbf{v}^{t+1}$ *to produce the initial state for the next time step. The perturbation is sampled from a distribution about* $\Delta\mathbf{v}$.

Most optimization techniques attempt to maintain plausibility by minimizing a scalar metric integrated over the entire simulation. For example, Chenney and colleagues [CF00] define the *plausibility function* as the product of Gaussian distributions over surface normals during collision events,

$$p \propto \prod_i e^{-(\theta_i/10)^2/2} \tag{4.1}$$

where θ_i is the angular displacement (in degrees) of the normal from its unmodified value (see Figure 4.2, left). Popović and colleagues minimized a weighted sum of the control parameters $\mathbf{u}$,

$$\mathbf{u}^T\mathbf{M}\mathbf{u} \tag{4.2}$$

where $\mathbf{u}$ is expressed using the exponential map [Gra98]. The effect of a metric that takes a sum or a product over a sequence of perturbations is to favor simulations that have only a few perturbations over those that perturb at each collision event. Such a metric can prevent a sequence of "unlikely" collision events from occurring. However, we are not aware any specific evidence that a series of smaller perturbations will appear particularly less likely than a single one, except insofar as it gives viewers more opportunities to notice that something may be amiss. Barzel and colleagues [BHW96] suggest one possible example: if an object takes a series of odd bounces that are all biased in the same direction, the observer might conclude that things have been somehow "rigged" (especially when look at dice or a roulette wheel). If we require each impulse to be selected independently the probability of choosing an entire sequence of nearly-identical independent perturbations will be the product of the individual probabilities and for a sufficiently-large sequence the product falls off exponentially.

Our own hypothesis is that the primary function of the plausibility metric is to prevent obviously bad bounces. Certainly, the use of a quadratic norm like (4.2) tends to penalize outliers more than an L^1 norm would; the Gaussian in (4.1) is even stronger, as events more than three standard deviations from the norm become all but impossible. For our own plausibility function, we use an L^∞ norm,

Plausibility condition: If each collision is individually plausible according to the criteria developed by O'Sullivan and colleagues [ODGK03], then the entire animation is considered plausible.

We limit velocity perturbation magnitudes to a fraction α of the original instantaneous collision impulse (see below), and in our system $\alpha = 0.1$.

Note that while our plausibility criterion is based on experiments performed by O'Sullivan and colleagues [ODGK03], the scenarios we will be dealing with are considerably more general than the single-collision gravity-free cases treated in that work. More experiments would be necessary to determine whether people are more sensitive when the geometry is more complicated or even an articulated character, although Reitsma and colleagues have some results in this area [RP03, RAP08]. On the positive side, it seems likely that people would be less likely to notice errors in crowded, distracting scenes, but this naturally depends on where their attention is focused [O'S05]. The final arbiter in this, however, is the user, and we certainly make no claim that our plausibility metric does anything beyond culling away poor-quality results.

4.1.2 Applying perturbations

The instantaneous state of a rigid body i is specified completely by its position $\mathbf{x}_i$ (assumed to specify the body's center of mass), its orientation Θ_i (commonly represented using quaternions), its linear velocity $\mathbf{v}_i$ and its angular velocity ω_i. A complete simulation of a collection of objects consists of a sequence of states as a function of time, $\mathrm{s}^t = \{\mathbf{x}_i^t, \Theta_i^t, \mathbf{v}_i^t, \omega_i^t\}$, for objects $i = 1 \ldots n$. We assume that we have a simulator capable of taking a collection of rigid body states s^t and producing the states $\mathrm{s}^{t+\Delta t}$. We note that, for the purposes of this discussion, we do not assume any further knowledge about the simulator (e.g., contact information) which should make this approach very flexible. According to our definition of plausibility, we can treat each impact event independently; we therefore suggest the following simple algorithm for sampling plausible simulations. We define the instantaneous change in velocity $\Delta\mathbf{v}_i$ as

$$\Delta\mathbf{v}_i = (\mathbf{v}_i^{t+\Delta t} - \mathbf{v}_i^t)/\Delta t \tag{4.3}$$

and analogously, for angular velocity,

$$\Delta\omega_i = (\omega_i^{t+\Delta t} - \omega_i^t)/\Delta t \tag{4.4}$$

Suppose we have generated the simulation up to state s^t. We use our solver to generate the state $\mathrm{s}^{t+\Delta t}$. The result of our algorithm will be the perturbed state $\tilde{\mathrm{s}}^{t+\Delta t}$. For each object in the scene, we examine the changes in linear and angular velocities, $\Delta\mathbf{v}_i$ and $\Delta\omega_i$. We look for "significant" impacts by comparing $\Delta\mathbf{v}_i$ and $\Delta\omega_i$ with two thresholds, $C_\mathbf{v}$ and C_ω.

Suppose that $||\Delta\mathbf{v}_i|| > C_\mathbf{v}$ (angular velocity changes are handled identically). According to the criterion we defined above, we can perturb the post-impact velocity $\mathbf{v}_i^{t+\Delta t}$ by up to α times the velocity change magnitude $||\Delta\mathbf{v}_i||$. To do this, we first sample a perturbation direction $\hat{\mathbf{d}}$ from a $\cos^n$ distribution (commonly used in the Phong lighting model) about the original impulse direction $\Delta\mathbf{v}_i$. We then sample a perturbation magnitude m uniformly from the range $[-\alpha\,||\Delta\mathbf{v}_i||\,, \alpha\,||\Delta\mathbf{v}_i||]$ (see Figure 4.2, right). We add $m\hat{\mathbf{d}}$ to $\mathbf{v}_i^{t+\Delta t}$ to produce the new post-impact state $\tilde{\mathrm{s}}^{t+\Delta t} = \{\mathbf{x}_i^{t+\Delta t}, \Theta_i^{t+\Delta t}, \mathbf{v}_i^{t+\Delta t} + m\hat{\mathbf{d}}, \omega_i^{t+\Delta t}\}$, which is then used as input to the solver in the next time step. This continues until all the objects in the scene come to rest or until a user-specified end time has been reached.

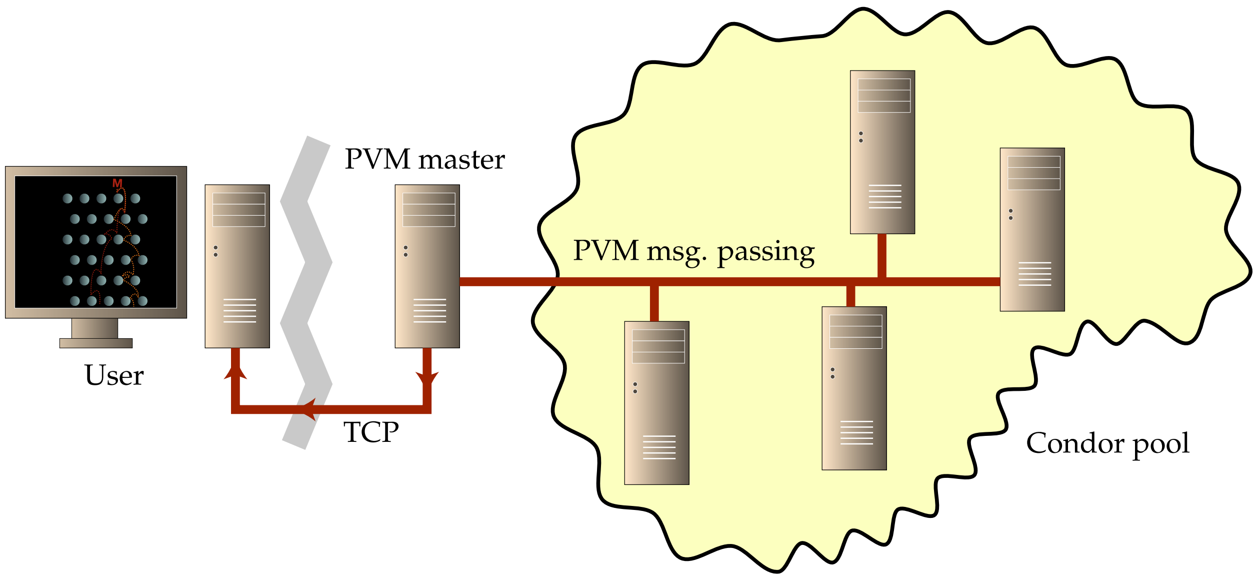


Figure 4.3: Streaming examples using Condor-PVM: *Because Condor-PVM is not supported on all platforms, examples must be passed back to the PVM master before being send back to the user via a TCP socket connection.*

4.2 Cluster-based sampling

Because each example motion is computed independently, we can easily parallelize this computation across many machines. In our case, we use a 24-node cluster; objects and initial conditions are transferred to each compute node and the resulting animation examples are transferred back as they are computed.

The initial implementation of this used the Condor project's Condor-PVM system, which is built on the Condor framework for distributed computing [Lit87]. Unlike the standard Condor environment, Condor-PVM allows for "opportunistic computing;" that is, the Condor system supplies compute nodes dynamically as they become available rather than holding the job until a precise pre-specified number of machines are free. As a result, the number of available machines may change dynamically over the course of the computation, mandating the ability to handle both the addition and removal of machines from the compute pool. The advantage of this framework is the ability to take advantage of machines that were otherwise unavailable to us, such as machines in the undergraduate labs, without the need to write code to ensure that we weren't interfering with the students who had priority. This has similarities to the *grid computing* paradigm that has become popular in other domains [LSSP02].

The Condor-PVM application programming interface (API) is based on the well-known Parallel Virtual Machine model [GBD$^+$94] (PVM) which is essentially a message-passing framework. In theory, basic PVM code could be ported to Condor-PVM with little modification, although the risk that machines may disappear from the compute pool without warning mandates that we take extra care when coding. Unfortunately, Condor-PVM's cross-platform support is limited and it includes no support for the Microsoft Windows operating system. As a result, we could not use PVM's transmission mechanisms to send data directly to the user's machine, mandating a two-stage transfer process. First, motion data was computed on machines in the Condor pool and transferred using PVM message-passing back to the master machine, at which point it was sent to the user's machine via a simple TCP socket connection. In the end, however, Condor-PVM proved unsuitable for our purposes. First, the Condor scheduler seemed better tuned for batch jobs than for interactive processes and would frequently stall for minutes or even hours before starting jobs up, even when machines were available. Second, while most PVM calls have non-blocking alternatives, the `pvm_spawn()` call appears to block forever, even in the opportunistic Condor-PVM environment. This meant that if a machine were removed from the pool after we received the `HOSTADD` message notifying us of the host's availability but before we could call the `pvm_spawn()` command that started a sampling job, the system would block awaiting a response from a machine that was no longer listening, and computation would cease.

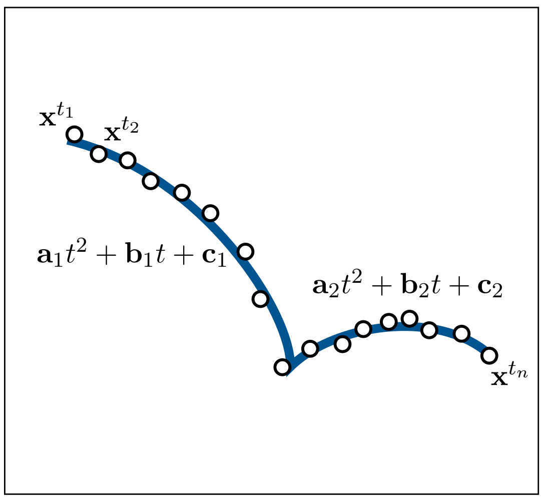
As a result, we were forced to discard our Condor-PVM-based implementation and fall back to a far simpler version that used only TCP sockets. In this scheme, jobs are started as server processes on individual cluster nodes. The user's machine then makes a direct TCP connection each available sampling node; the scene description is transferred from the user's machine to the sampling node, and motion data is transmitted back as soon as it is computed. Motion data is sent compressed (see §4.3) and using a binary format, which is straightforward provided byte ordering is respected. For robustness, we use a CRC checksum [PB61] to ensure that each motion segment is received correctly; although TCP's own checksum mechanism makes this redundant, it adds little to bandwidth requirements and is useful for debugging. Further details of how this is handled on the user's end can be found in §4.4.4.

4.3 Compressed in-core representation

A single animation consisting of several hundred objects can take a significant amount of memory to store and transmit, and we want to store hundreds of these examples in core so that they can be retrieved, displayed, and queried quickly. If we maintain sufficiently high quality in the compressed version, it will save us from needing to maintain an extra copy of the data on disk, although this is also an option. To this end, we have developed a simple compression scheme which significantly reduces the storage cost of each animation while placing an absolute per-frame, per-object limit on the amount of error introduced by the approximation.

Rigid-body motion is generally quite smooth between collision events. We have therefore chosen to use splines to represent the motion. An added benefit of this approach is that we can store motion at 120 frames per second, which will give us enough temporal resolution to render correct motion blur for impacts.

We compress the object's position and rotation components separately. For position, we simply fit piecewise-quadratic splines $f(t) = \mathbf{a}_i t^2 + \mathbf{b}_i t + \mathbf{c}_i$ to the motion of the object's center of mass,


$\mathbf{x}^{t_1}$
$\mathbf{x}^{t_2}$
$\mathbf{a}_1 t^2 + \mathbf{b}_1 t + \mathbf{c}_1$
$\mathbf{a}_2 t^2 + \mathbf{b}_2 t + \mathbf{c}_2$
$\mathbf{x}^{t_n}$

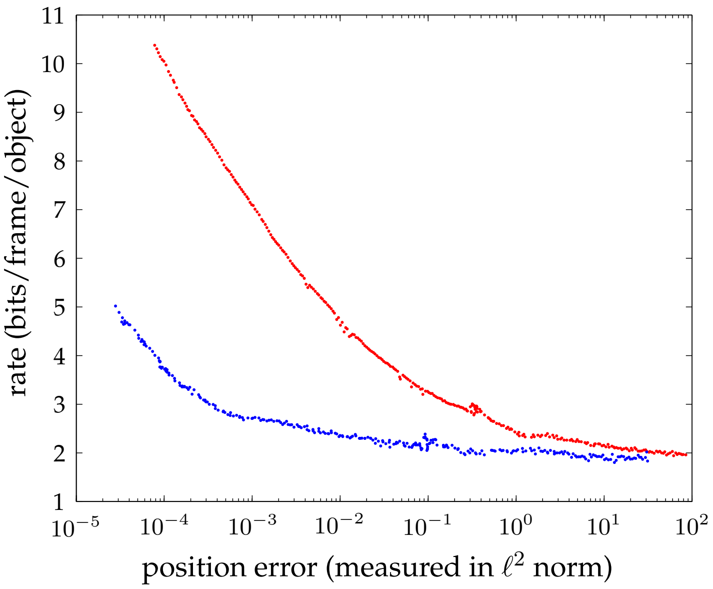


Figure 4.4: Left: *For compression, we fit piecewise quadratic splines to object motions.* **Right:** *Rate-distortion curve for the position component in our compression scheme; the blue curve corresponds to a collection of objects in free flight while the red curve corresponds to an articulated character. Notice that storage cost for the articulated character is significantly higher.*

with $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^3$, as shown in Figure 4.4, left. To improve the compression ratio, we then quantize the spline coefficients. Adjacent coefficients are differenced, and the resulting residuals are stored using an encoding scheme known as Rice codes [Sal04] that uses fewer bits for smaller numbers. However, there are a few details that make a sizable difference in the compression.

- First, we know that the paths are C^0 continuous, so we can assume that each piecewise quadratic segment starts where the previous one ends. This allows us to discard the constant term $\mathbf{c}_i$ for all but the first segment, reducing space usage by 33%.

- Quantizing the quadratic coefficient $\mathbf{a}_i$ produces sizable errors, because t^2 grows much faster than t which magnifies quantization error. However, the quadratic coefficient of translational motion tends to be constant during free flight (this would not be the case, however, in the presence of aerodynamic drag). We can therefore use a single quadratic coefficient for the entire animation; in most cases this will be $h^2 g/2$ for gravitational constant g and time step size h. This has the added benefit of reducing the quadratic coefficient to a single bit, dropping space usage by another 33%.

- Finally, provided air damping is minimal, we only need to retain the quadratic coefficient for the vertical component of the object's position.

To maintain quality, we place a limit e_{max} on the ℓ^∞ norm of the error; this will guarantee that no single frame has high error. We use least-squares (specifically, a QR decomposition) to fit a polynomial segment to the object positions $[\mathbf{x}^{t_1}, \mathbf{x}^{t_2} \ldots \mathbf{x}^{t_n}]$ recorded at times $[t_1, t_2 \ldots t_n]$. Although we tried methods that minimized the ℓ^∞ norm using linear programming [Spa91], we found that

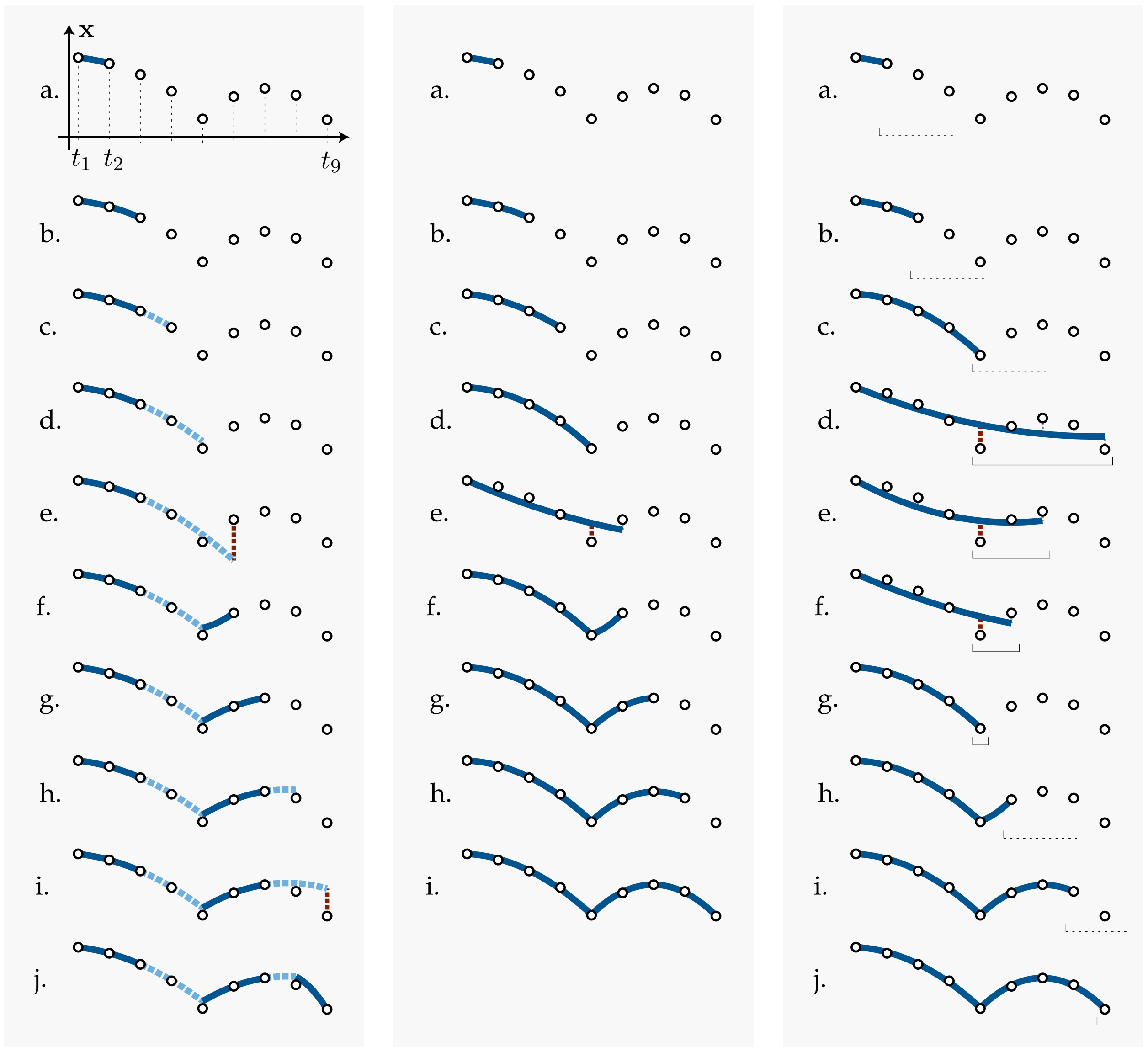


Figure 4.5: Comparison of greedy algorithms for fitting splines *to the 1-dimensional data samples* $[\mathbf{x}^{t_1}, \mathbf{x}^{t_2} \ldots \mathbf{x}^{t_9}]$. **Left:** *The algorithm described in Figure 4.6; we fit using least-squares to the first k points in each segment* **(a-b)**; *here $k = 3$) and for further points simply extend the previously computed spline* **(c-d)** *until the error is too large* **(e)**. *This avoids the $\mathcal{O}(n)$ cost of fitting spline segments for $n > k$, but if k is too small we may compute a poor-quality fit* **(i)**, *meaning that extra spline segments will be needed* **(j)**. **Center:** *An alternative approach is to re-fit the spline for each subset of the points; this produces a better fit* **(i)** *but has $\mathcal{O}(n^2)$ cost.* **Right:** *We can address the $\mathcal{O}(n^2)$ runtime using a binary search; here we use exponential growth to get an upper bound on the range of points to fit* **(a-d)** *and then use binary search to find the exact value* **(e-g)**. *Unfortunately, because this algorithm typically overshoots by a factor of two* **(d)**, *it is considerably slower in practice than our algorithm despite having the same asymptotic performance.*

least-squares worked well enough and had substantially lower run-time cost. Selecting where to place transitions between splines is a harder problem. While it is possible to compute the optimal fit using dynamic programming [NH98, HHH97], these methods are expensive, and we use the much simpler greedy algorithm described in Figure 4.6 in our work (see also Figure 4.5, left for a visualization). Essentially, the algorithm fits a polynomial to the first $k = 10$ points and thereafter extends the current segment until it reaches a time t_{end} at which the fitting error exceeds the maximum allowed error,

$$\left|\left|\mathbf{x}^{t_{end}} - (\mathbf{a}t_{end}^2 + \mathbf{b}t_{end} + \mathbf{c})\right|\right| > e_{max} \tag{4.5}$$

The integer k represents a trade-off between compression quality and speed. If we set $k = \infty$, then the system will recompute the curve fit each time it checks a new point $\mathbf{x}^{t_i}$ against condition (4.5) (see Figure 4.5, center). Using QR, the cost of computing the least-squares fit to n points using r parameters is $\mathcal{O}(nr^2)$ [GL96]. If we do this for each point $[\mathbf{x}^{t_1}, \mathbf{x}^{t_2} \ldots \mathbf{x}^{t_n}]$, the cost will be

$$\sum_{i=1}^{n} ir^2 = \frac{n(n+1)}{2} r^2 \tag{4.6}$$

This produces higher quality results than our algorithm but is prohibitively expensive for large n. To eliminate the quadratic cost, we can use a binary search to find the endpoints of the fitted range (see Figure 4.5, right). During the expansion phase of the algorithm (steps a-d in Figure 4.5) we will fit to 2 points, then 3 points, then 5, then 9 points, up to n. This results in the following sum,

$$\sum_{i=1}^{\lceil \log n \rceil} r^2 2^i \approx 2 \cdot 2^{\lceil \log n \rceil} r^2 = \mathcal{O}(nr^2) \tag{4.7}$$

This is asymptotically linear in n, but this algorithm is measurably slower than our final algorithm (Figure 4.6) due to its higher constant terms.

Although it is possible to construct higher-order splines on rotation spaces using quaternions [Sho85], we found a piecewise linear approximation to be sufficient. Let $[\Theta_i^{t_0}, \Theta_i^{t_1} \ldots \Theta_i^{t_n}]$ be a sequence of rotation matrices. Using the logarithm map [Gra98], we can represent the matrix $\Theta_i^{t_n}(\Theta_i^{t_0})^{-1}$ by the matrix exponential $e^{[\omega]}$, where $[\omega]$ is the skew-symmetric matrix [KEP05]

$$[\omega] = \begin{pmatrix} 0 & -\omega_2 & \omega_1 \\ \omega_2 & 0 & -\omega_0 \\ -\omega_1 & \omega_0 & 0 \end{pmatrix} \tag{4.8}$$

We can approximate intermediate rotations as $\Theta_i^{t_i} \approx e^{[(t_i - t_0)\omega/(t_n - t_0)]}\Theta_i^{t_0}$. The algorithm in Figure 4.6 is essentially unchanged, except that we replace the FIT-SEGMENT function with the algorithm just described, and our norm is the quaternion argument of the rotational error $(\Theta_i^{t_i})^{-1} e^{[(t_i - t_0)\omega/(t_n - t_0)]}$.

```
FIT-SEGMENT($\mathcal{F}, t_{start}, e_{max}$)
 1  $t_{end} \leftarrow t_{start} + 1$
 2  $(\mathbf{a}, \mathbf{b}, \mathbf{c}) \leftarrow (\mathbf{0}, \mathbf{0}, \mathbf{0})$
 3  while $t_{end} < \text{SIZE}(\mathcal{F})$
 4    switch
 5      case $t_{end} - t_{start} < 10$ :
 6          $(\mathbf{a}', \mathbf{b}', \mathbf{c}') \leftarrow$ FIT-SPLINE($\mathcal{F}[t_{start} \ldots t_{end}]$)
 7          $e \leftarrow \max_{[t \in t_{start}, t_{end}]} \left|\left|\mathcal{F}[t] - (\mathbf{a}'t^2 + \mathbf{b}'t + \mathbf{c}')\right|\right|$
 8      case default :
 9          $e \leftarrow \left|\left|\mathcal{F}[t_{end}] - (\mathbf{a}t_{end}^2 + \mathbf{b}t_{end} + \mathbf{c})\right|\right|$
10    if $e > e_{max}$
11      then return $(\mathbf{a}, \mathbf{b}, \mathbf{c}), t_{end}$
12    $(\mathbf{a}, \mathbf{b}, \mathbf{c}) \leftarrow (\mathbf{a}', \mathbf{b}', \mathbf{c}')$
```

Figure 4.6: Fast, greedy algorithm for fitting spline segments: *For performance reasons, we only perform a full least-squares spline fit (using QR decomposition) for the first 10 points in any spline segment. The algorithm takes as input the series of object positions $\mathcal{F} = [\mathbf{x}^{t_1}, \mathbf{x}^{t_2}, \ldots \mathbf{x}^{t_n}]$, the start time of the current segment (which is also the end time of the previous segment), and the maximum allowed ℓ_∞ error e_{max}, and produces as output the segment length $t_{end} - t_{start}$ and the spline coefficients $\mathbf{a}$, $\mathbf{b}$, and $\mathbf{c}$. Spline errors are computed with respect to the quantized coefficients to get a more accurate estimate of the display error.*

4.4 Interactive browsing

Once we have computed a sufficient number of example animations, we need to provide ways for the user to interact with and select among the possibilities. We provide two basic modes of interaction: queries and metrics.

4.4.1 Spatial queries

Many requests the user could make, such as “make sure the character falls all the way down the stairs” or “have the car flip off the bridge and into the water,” can be satisfied using simple spatial queries. We model our query interface on the work of Hochheiser and Shneiderman [HS04]. Their approach was originally developed for use with time-series data, and we will summarize it here.

Suppose we have a large number of functions $f_1(t), f_2(t), \ldots f_n(t)$, defined for $t \in [0, t_{\text{final}}]$. The user would like to explore this data set to find specific trends, e.g., functions $f_k(t)$ that start low at $t = 0$ and finish high at time $t = t_{\text{final}}$. Rather than specifying this query using a text query interface such as SQL, users of this system can simply draw boxes of the form $[t_1, t_2], [y_1, y_2]$, and the system returns all functions f_k such that $y_1 \leq f_k(t) \leq y_2$ for all $t_1 \leq t \leq t_2$. Feedback is immediate, and the user can specify more complex queries using the conjunction of simpler queries. Sherbondy and colleagues [SAM$^+$04] applied a similar paradigm in the context of diffusion tensor imaging, using queries to select among pathways in the brain.

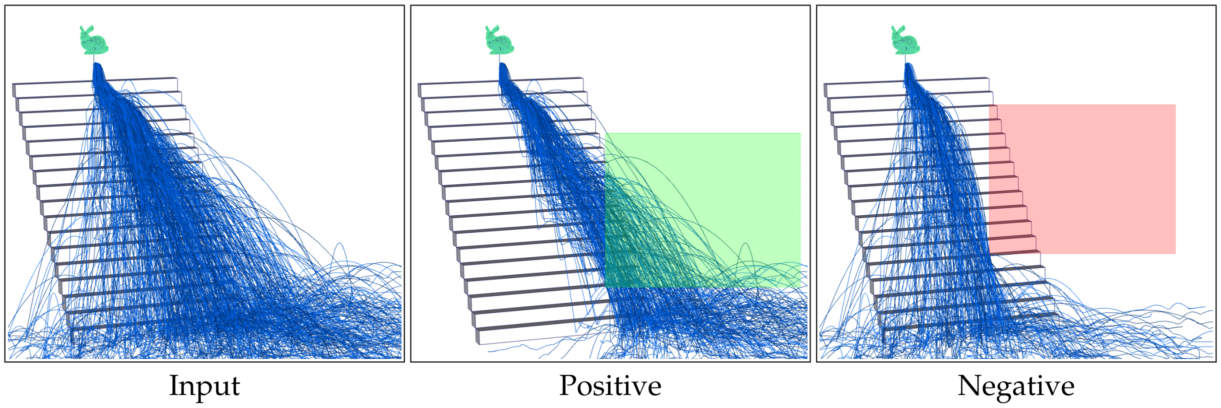


Figure 4.7: Spatial queries *in screen space are used to include and exclude simulation examples.* **Left:** *All input simulation examples for a single body example.* **Center:** *Positive simulation constraint boxes (green) include simulation worlds with motion inside.* **Right:** *Negative simulation constraint boxes (red) exclude simulation worlds with motion inside.*

Our queries will be similar but will operate on the center-of-mass trajectories of simulated objects, as shown in Figure 4.7. Focusing on the center of mass allows us to easily visualize large numbers of paths without overloading users; however, it means that users can only visualize rotations by selecting a particular path and playing back the motion. Popović and colleagues [PSE$^+$00] addressed this by showing the paths of two distinct points on the object surface, but when many paths are being displayed the user cannot easily determine which pairs of points are in correspondence. The simultaneous visualization of many rotational motions is an interesting area of future work.

To work with simulation data, we modify the method of Hochheiser and Shneiderman slightly. First, to keep the interface as simple as possible, queries are drawn in screen space. This lacks the precision of the 3D query prisms introduced by Sherbondy and colleagues, but allows for much faster interaction on the user's part. The resulting query volumes correspond to frustums in world space. Queries can be either positive or negative, where positive queries return only simulations that pass through the query volume and negative queries return only simulations that do not pass through the selected volume. We also by default discard temporal information about object motion when evaluating queries; that is, queries are satisfied by simulations that pass through the query region at any point during the object's motion. One could override this behavior by painting an appropriate region on the time slider.

To improve the descriptive power of queries, we allow them to be evaluated for any user-chosen subset of the scene objects. Thus, queries such as "require that one of these objects pass through region R" can be easily satisfied by selecting the particular group of objects and drawing the desired query.

4.4.2 Accelerating query handling

In order for our system to be useful, it is vital that the interface be responsive even when processing massive data sets. Rapid evaluation of queries is particularly important, as their usefulness depends largely on the user's ability to quickly adjust the queries depending on the responses retrieved from the database.

The first method for ensuring that queries can be answered quickly is to retain an uncompressed and down-sampled version of each object's path in memory at all times. We can visualize paths and evaluate queries using this low-resolution version, but use the more accurate compressed representation when displaying actual object motion.

To evaluate whether an individual path satisfies a query, we need to quickly check whether the path intersects the frustum defined by the query. A variety of acceleration structures could be used here, but because paths have no volume, many forms of bounding geometry represent them poorly. Sphere trees, for example, proved to be slower than a simple brute force check in our tests. We found that oriented bounding boxes (OBBs) provide a good balance of efficiency and representational power. OBB-frustum tests can be made quite fast, as described by Assarsson and Möller [AM00b]. To reduce the memory overhead of the structure, we restrict the tree depth to be at most three, and piecewise linear segments of the path are stored in the leaf nodes. We note that we can always reconstruct the entire path from its tree representation, so the OBB tree representation of the path obviates the need to store the down-sampled path separately. In our tests, using this hierarchy speeds up query evaluation by a factor of about three.

In addition, these queries are easy to parallelize. In our implementation, when the user initiates a query, the system starts up a single thread for each available processor (commonly 2 on today's modern dual-core machines, but most platforms provide the means to query the system for available processors and memory). This ensures an additional 2x speedup. The result is a query system that is fast enough in practice (a few seconds at most per query) for the simulation sizes that we were able to run using the ODE simulator and the available cluster; larger, hardware-accelerated simulations that we hope to see in the future may necessitate more work on accelerating queries.

If the user discovers that a previously applied constraint is too restrictive, she may want to remove that constraint (see Figure 4.8). In our system, there are two ways to remove a constraint. The most recent constraint can be eliminated using the Undo stack; this is useful as it is common to draw a constraint and immediately realize that the result is not what was desired. For older constraints, the user must first select the constraint. To facilitate this, we draw the frustum of the constraint in space, using fractional alpha to ensure that it doesn't interfere with other aspects of the system (see Figure 4.9). The user can then select the constraint by simply clicking on it (to check which constraint the user has chosen requires only a simple ray-frustum intersection test).

Allowing the user to remove arbitrary constraints, however, means that if we only maintain a record of the current "active" paths, then removing a single constraint means we will need to iterate through all the paths that might have been affected by the removed constraint and check them against each of the remaining constraints. To handle this, we instead store alongside each

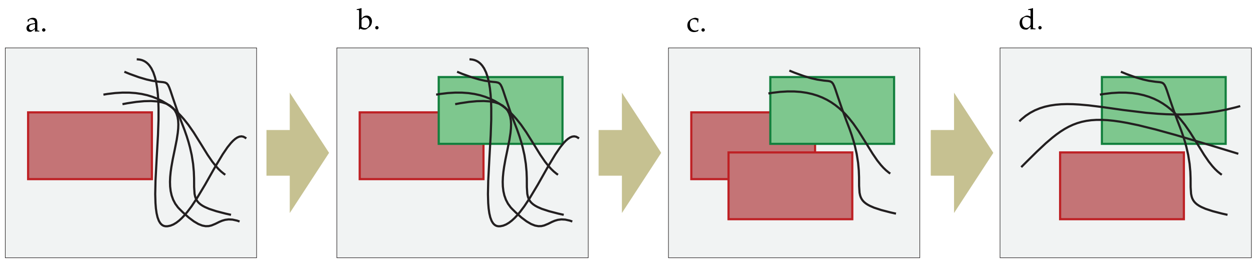


Figure 4.8: Removing a constraint: *Because constraints are combined using the logical* and *operator, it is easy to over-constrain the system and eliminate virtually all the paths from the system. To counteract this, we allow users to easily remove constraints from the system. Here, the user adds several negative* **(a, c)** *and positive constraints* **(b)** *but decides that the resulting set of simulations* **(c)** *is not sufficient; she then removes one of the negative constraints so that the system will show her additional paths* **(d)**.

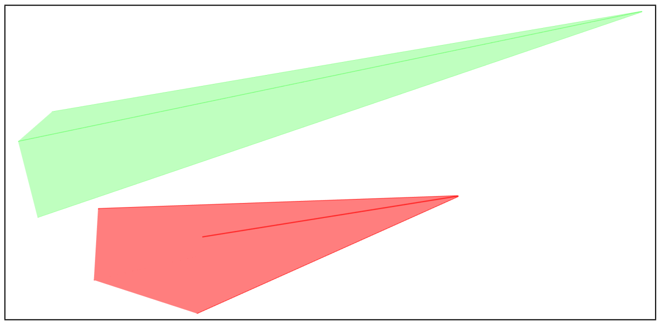


Figure 4.9: Constraint frustums: *Constraint frustums are displayed in the browser window and users can click on them to add or remove constraints.*

constraint a bitset [Ben99] in which bit i is set to indicate that example motion i satisfies the given constraint (this assumes that each motion is assigned a unique integer identifier). Using the bitset representation, intersections of sets can be computed very efficiently. If a constraint is removed, we need only step through the remaining constraints computing the conjunction of each associated bitset. This eliminates the need to take the expensive step of re-evaluating the constraints spatially.

Our browsing interface could be extended in several directions. Currently constraints apply only to center of mass positions rather than swept volumes, which leads to unintuitive results, e.g., the user draws a constraint expecting that the object will never pass through a region, and is surprised to discover that the constraint does not apply to the object's corner. Fixing this would be a straightforward application of collision handling techniques, although it might slow down query processing somewhat. More generally, we could make the constraints have limited time scope, for example, or design some form of orientation constraint to make problems like getting the SIGGRAPH letters upright easier to solve.

4.4.3 Ranking metrics

In many cases, the constraints the user places on the result will be very loose, and as a result many example simulations will satisfy them. If there are only a few such examples, the user can easily sort through them by iterating through each individually. If there are a large number, however (> 100), it will be burdensome for the user to view them all. In this case, we provide *simulation metrics* to aid in the search.

The intuition behind these metrics is that the user may wish to find the simulation satisfying the specified constraints that is "most exciting" or "least distracting," or perhaps the user has a specific frame budget for the simulation to come to rest. We suggest a number of metrics here, but there are many other possibilities that could be explored.

- **Angular velocity:** One measure of how dynamic a simulation looks is how much objects rotate, which we compute by integrating angular velocity across the entire simulation.
- **Running time:** We measure the amount of time it takes a specific object or set of objects to come to rest. This can help to keep simulations within frame budgets, or to look for the simulation that keeps a motorcycle upright and rolling as long as possible.
- **Collisions:** We can count the number of collisions in the scene using the same thresholding technique we used to determine when to apply impulse perturbations; this can produce interesting results such as a character hitting every stair on the way down.
- **Orientation:** Our simple screen-space constraints do not give us any control over orientation. One way to compensate for this is to add a metric that measures the difference between an object's final orientation and a particular goal orientation specified by the user. A more sophisticated variant would also allow the user to specify a time. Our preliminary implementation simply uses the object's starting orientation as the goal orientation, which was sufficient for our examples.

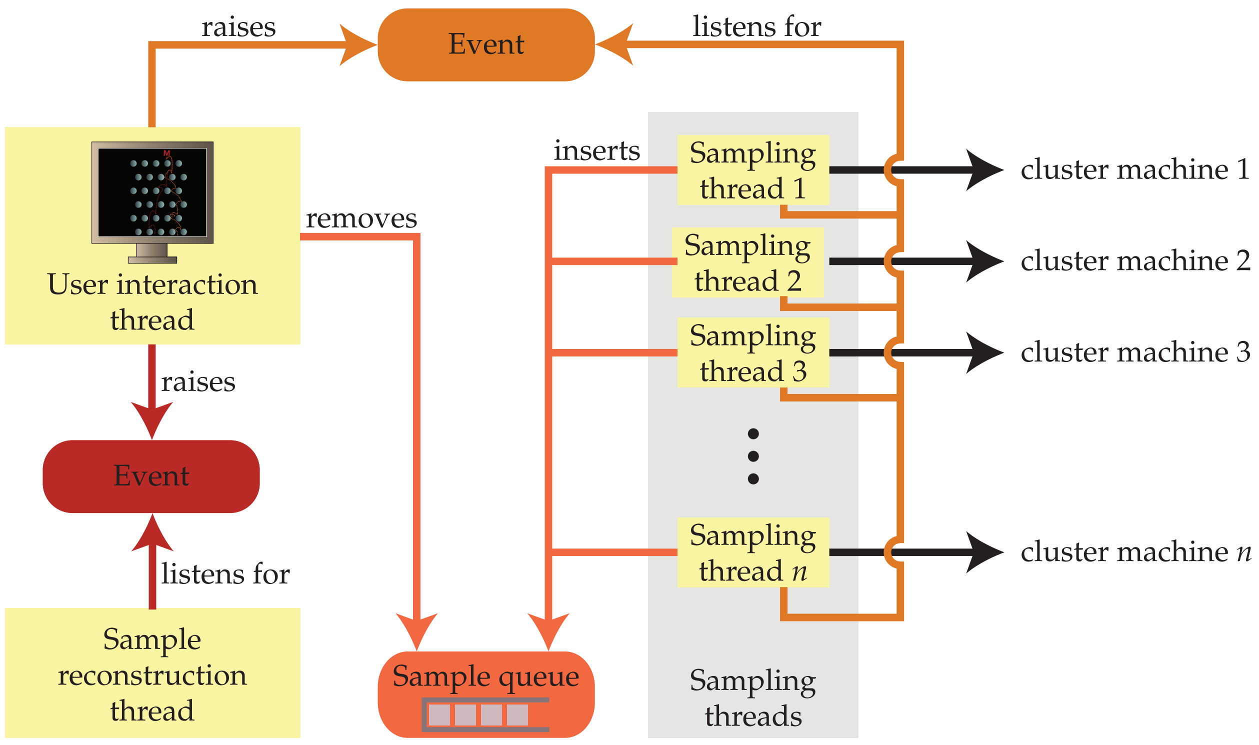


Figure 4.10: Threading setup: *To maintain interactivity, our system uses $n + 2$ threads, where n is the number of compute nodes in our cluster. One of the threads is responsible for handling all user interaction, one is responsible for decompressing motion, and the remaining n threads connect to the remote machines and read the motion that is computed remotely off the network.*

- **Satisfied constraints:** In many situations, the user may start out with a set of constraints that is not satisfied by any example. In this case, further refinement will be necessary to produce the desired result, but we should give the user some feedback on which paths to refine. One simple technique is to simply count how many constraints each example satisfies and sort using this. For this to be effective, we need to change the behavior of our viewer slightly to show all examples and not just the set that satisfy user constraints.

Relevant metrics can be computed on a per-object basis to allow for more precision; the user simply selects which objects should be measured. We can precompute the results of the metrics (except the "satisfied constraints" metric) for each object individually. When sorting, we simply look up the metric value for each object and take the sum/supremum as appropriate over all such values. The sorting is quite fast and the results are displayed in our "metric sorter" view, which is essentially identical to list boxes commonly found in modern GUI applications.

4.4.4 Implementation

There are a number of implementation details that affect the user experience. Keeping the system interactive at all times helps to minimize frustration and keeps users focused on the task at hand. To ensure this, the entire application is multithreaded, so less-urgent tasks can be run in the background while the user interaction thread remains responsive. Although cross-platform threading libraries are available (the Boost C++ Libraries include one [Kem03]), these libraries are limited to the small subset of functionality which is available on all platforms. Certain platform-specific threading primitives proved useful during the development of this project, so while the sampling aspects of the system can be run on any system, the interactive portion of the system is currently limited to the Microsoft Windows operating system.

wxWidgets, a user-interface (UI) library used throughout the project [SHC05], requires that all user interaction occur on a single thread (Figure 4.10, upper left). As is common in UI libraries, specific interactions are handled using callback functions. Besides the usual callbacks (mouse clicks, repaint events), wxWidgets also allows us to provide an idle handler which is called when no other events need handling.

In addition to the primary UI thread, we start up a moderate-priority background thread that is responsible for reconstructing selected paths from the compressed representation (Figure 4.10, lower left). This is necessary because decompressing paths involving more than a few dozen objects may take several seconds, and this must be done each time the user selects a new path. Performing this decompression in the foreground thread would result in unacceptible delays each time the user clicks on a path; this would be especially frustrating as selecting a single path out of thousands can be a somewhat error-prone process (this is a natural consequence of Fitt's law [Fit54] and the small screen-space distance that separates tightly packed paths). Therefore, when the user selects a path, the UI thread first displays motion data that is uncompressed but has extremely low temporal resolution (1 frame per second); this data is stored alongside the compressed path and is also used for rendering in the metric sorter view. The UI thread then raises an event notifying the sample reconstruction thread that a new motion has been selected, and the reconstruction thread drops any other tasks and begins decompressing the newly selected path. When decompression is completed, the low-resolution motion is replaced with the new motion (this can be done atomically using mutual exclusion locks) and the UI thread is notified to repaint itself.

As discussed in §4.2, motion data is computed on cluster machines and sent back to the user machine over a TCP connection. On the user's machine, each cluster node is assigned a single thread which is responsible for starting up and listening on the remote connection (Figure 4.10, right). As each example is received, it is placed in a queue that is read by the UI thread during the idle callback. Although this queue could be implemented using lockless data structures [Fra03], in our case we handle contention between threads using a basic mutual exclusion lock.

One important aspect of the system is that the UI thread needs to be able to communicate with the sampling threads to notify them that the user has performed certain actions, such as changing the scene or refining the current path. Scene changes need to be passed on to the cluster machines as quickly as possible so that they can begin computing motion using the new settings; otherwise,

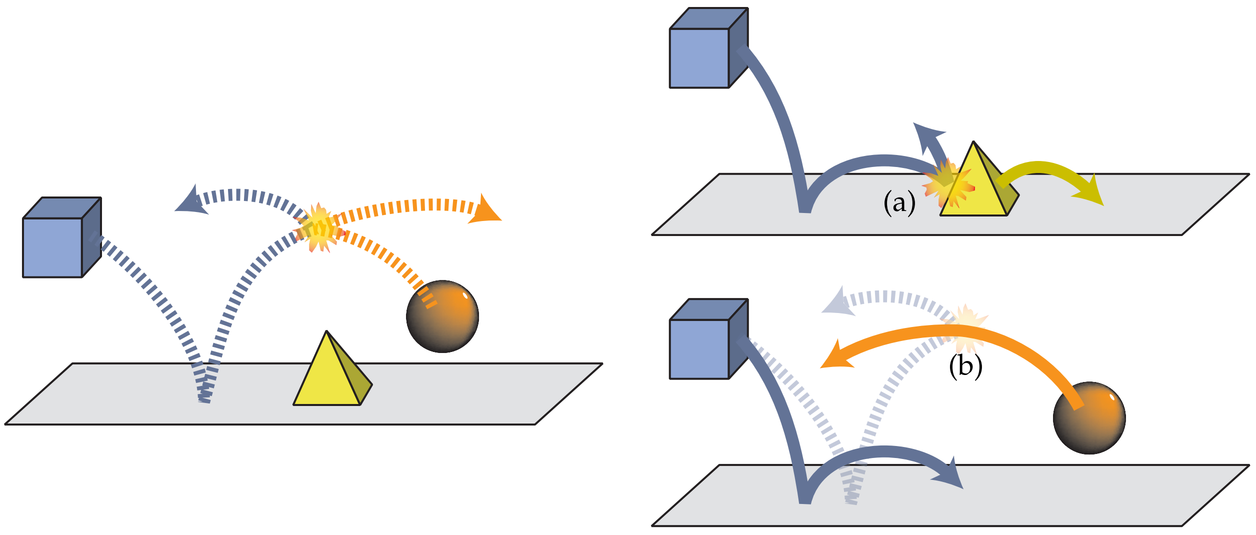


Figure 4.11: Refining object motion in the presence of collisions: *If we refine the original (dotted) path taken by the blue box (***left***), we must ensure that other objects in the scene still have physically valid motion. Specifically, if the new motion interacts with other objects in the scene (as at point* **(a)**, **right***), we must simulate new motion for these objects; additionally, if in the newly computed motion the box fails to interact with some object that it interacted with in the original motion (point* **(b)***, bottom), we must compute new motion for this object as well.*

the user will notice a delay before new samples start appearing in the UI. To ensure that the sampling threads do not stall indefinitely waiting for data to arrive over the network connection, we use non-blocking I/O. A separate channel (an Event object in the Windows API object or an unnamed pipe in Posix) is reserved for the UI thread to communicate with each sampling thread (in Figure 4.10 these are combined for simplicity in representation), and each sampling thread is required to simultaneously listen on both the remote socket and the channel connecting it to the UI thread. On Linux, a `select()` loop allows listening on both channels; on Windows, overlapped I/O and the `WaitForMultipleObjects()` system call accomplish the same thing.

4.5 Refinement

One drawback of the approach described thus far is that for each example the entire simulation is computed before being shown to the user, who must respond with a binary accept/reject decision whether one of the displayed examples is acceptable. For scenes with large numbers of objects or long running times, the probability that any one simulation will match all the user's objectives can be prohibitively small. It would be useful to be able to take simulations which are mostly correct and fix the parts which are not; we call this process "refinement."

To indicate that he wants to refine the path of a particular object, the user simply selects the object i and draws a box (in screen space, as before) indicating which part of the path he wishes to change. We find the smallest time t_{min} such that the object's position $\mathbf{x}_i^{t_{min}}$ (as defined by the selected example) is contained within the frustum defined by the box, and begin computing new examples which start at time t_{min} with state $\mathbf{s}^{t_{min}}$. If he wants, the user can select more than one object, in which case we simply compute the smallest t_{min} such that the center of mass position $\mathbf{x}_i^{t_{min}}$ of any of the selected objects is contained within the frustum.

Because the user specifies precisely the set of objects to refine, we attempt to keep the paths of other objects in the scene unchanged. In certain cases, however, physical accuracy will force us to compute new motion for these other objects. Specifically, if one of the refined objects interacts with another object, we will need to compute new dynamics in which the other object responds in a physically accurate way (see Figure 4.11(a)). Moreover, if the refined object interacted with another scene object in the original motion but does not in the newly computed motion, we must compute new dynamics that accurately reflect this (see Figure 4.11(b)).

To ensure that we handle these scenarios correctly, we maintain three sets of objects during simulation:

$\mathcal{O}_a$ **Actively simulated objects**: at the beginning of simulation, this consists of only those objects marked by the user for refinement.

$\mathcal{O}_i$ **Inactive objects** whose paths simply track the stored motion. Initially, this set consists only of objects *not* marked by the user for refinement, but as they interact with actively simulated objects we move them into $\mathcal{O}_a$ so that they respond realistically (Figure 4.11(a)).

$\mathcal{O}_g$ **Ghost objects**: for each object $i \in \mathcal{O}_a$, we add a corresponding "ghost object" i' to $\mathcal{O}_g$ whose motion tracks the original (pre-refinement) motion of i. This ensures that we catch objects that collided with i in the original motion but may no longer do so in the newly generated motion (see Figure 4.11(b)); these must be added to $\mathcal{O}_a$ to guarantee physical validity.

The algorithm for updating these sets is shown in Figure 4.12. Note that it requires slightly more information from the simulator than we needed in our original sampling algorithm: specifically, we need to detect all interactions between objects in the scene. Simulators commonly make this information available through some form of callback interface.

For objects in $\mathcal{O}_g$, we expand the collision geometry slightly to ensure that we catch interactions that might have been missed due to quantization errors in the stored motion data. Because we do not need to compute dynamics for objects in $\mathcal{O}_i$, refinement can be significantly faster than the original simulation for scenes whose simulation cost is not dominated by collision checking. Much as with the generation of the initial examples, we can compute the new motion on our cluster, and the fact that we only need to transfer the changed parts of the object motion cuts down on storage and bandwidth requirements significantly.

One useful extension to this technique can be developed by noting that there is nothing about the REFINE algorithm described above that necessitates that the sets $\mathcal{O}_a$ and $\mathcal{O}_g$ be in perfect correspondence. We could for example add a new object to the scene by simply inserting it into $\mathcal{O}_a$ at the beginning of the simulation. Similarly, if we wanted to remove an object i from the scene, we could add i' to $\mathcal{O}_g$ without adding i to $\mathcal{O}_a$ (or $\mathcal{O}_i$). This will ensure that the resulting mo-

```
REFINE($\mathcal{O}_a, \mathcal{O}_i, \mathcal{O}_g, t_{min}$)
1  $t \leftarrow t_{min}$
2  while Simulating
3     $C \leftarrow$ DETECT-INTERACTIONS($\mathcal{O}_a \cup \mathcal{O}_i \cup \mathcal{O}_g$)
4     for each $(i, j)$ in $C$
5        switch
6           case $i, j \in (\mathcal{O}_a \cup \mathcal{O}_i)$ :
7                 $\mathcal{O}_a \leftarrow \mathcal{O}_a \cup \{i, j\}; \mathcal{O}_g \leftarrow \mathcal{O}_g \cup \{i', j'\}$
8                 $\mathcal{O}_i \leftarrow \mathcal{O}_i - \{i, j\}$
9                 HANDLE-INTERACTION($i, j$)
10          case $i \in \mathcal{O}_i$ and $j \in \mathcal{O}_g$ : Symmetric case is identical
11                $\mathcal{O}_a \leftarrow \mathcal{O}_a \cup \{i\}; \mathcal{O}_g \leftarrow \mathcal{O}_g \cup \{i'\}$
12    SIMULATION-STEP($\mathcal{O}_a$)
13    COMPUTE-PERTURBATIONS($\mathcal{O}_a$)
14    for each $i$ in $\mathcal{O}_i$
15       $s^i \leftarrow$ LOAD-STORED-MOTION($i$)
16    for each $i'$ in $\mathcal{O}_g$
17       $s^{i'} \leftarrow$ LOAD-STORED-MOTION($i$)
```

Figure 4.12: Algorithm for correctly handling object interactions during motion refinement: *Starting with the initial set of active objects $\mathcal{O}_a$, we expand $\mathcal{O}_a$ to include other objects in the scene that interact either with objects in $\mathcal{O}_a$ or their corresponding objects in $\mathcal{O}_g$.*

tion accounts correctly for the removal of i's interactions with other objects. Changes to scene objects (e.g., modifying an object's starting velocity) can be interpreted as the combination of an object removal and an object addition, giving us wide latitude in the kinds of adjustments we can make to the scene even as we are browsing through available simulations. Users could decide that an explosion needs more shrapnel, for example, without changing the motion of most objects already in the scene. Such changes are demonstrated in the refrigerator toy example shown in the accompanying video.

4.6 Results

We applied our technique to several examples. Where appropriate, we will discuss the user strategies used in generating a particular example.

4.6.1 Refrigerator toy

We mocked up a popular physics toy where the goal is to roll a marble down a complicated path involving various types of chutes and rotating buckets (see Figure 4.13). Using our system it is easy to create such examples; we can continually add objects to the scene and place them to lie in the path of the marble. Sampling is used to find animations where, for example, the cup rotates

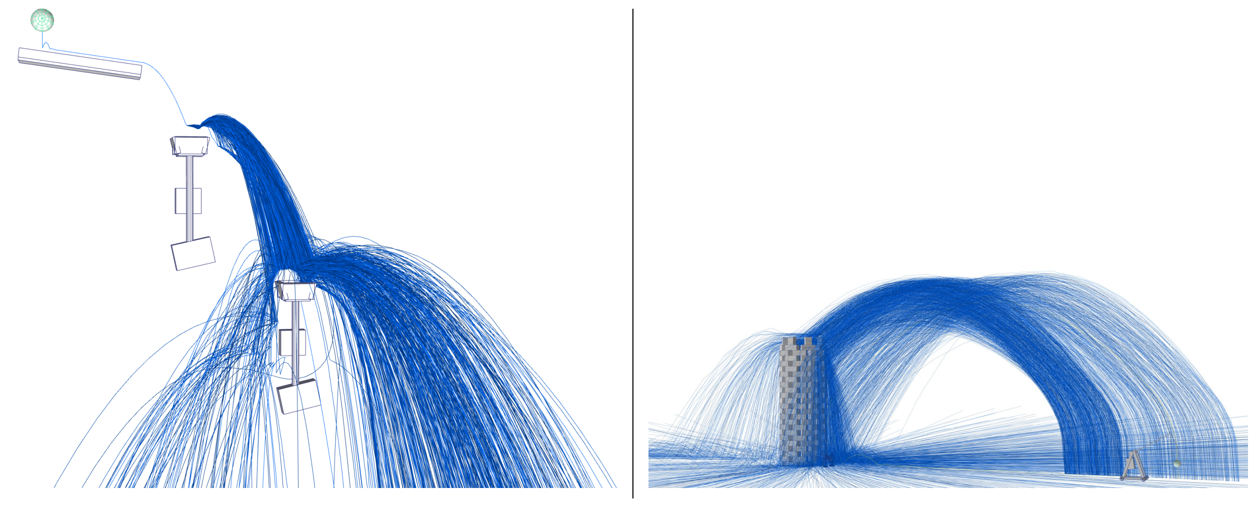
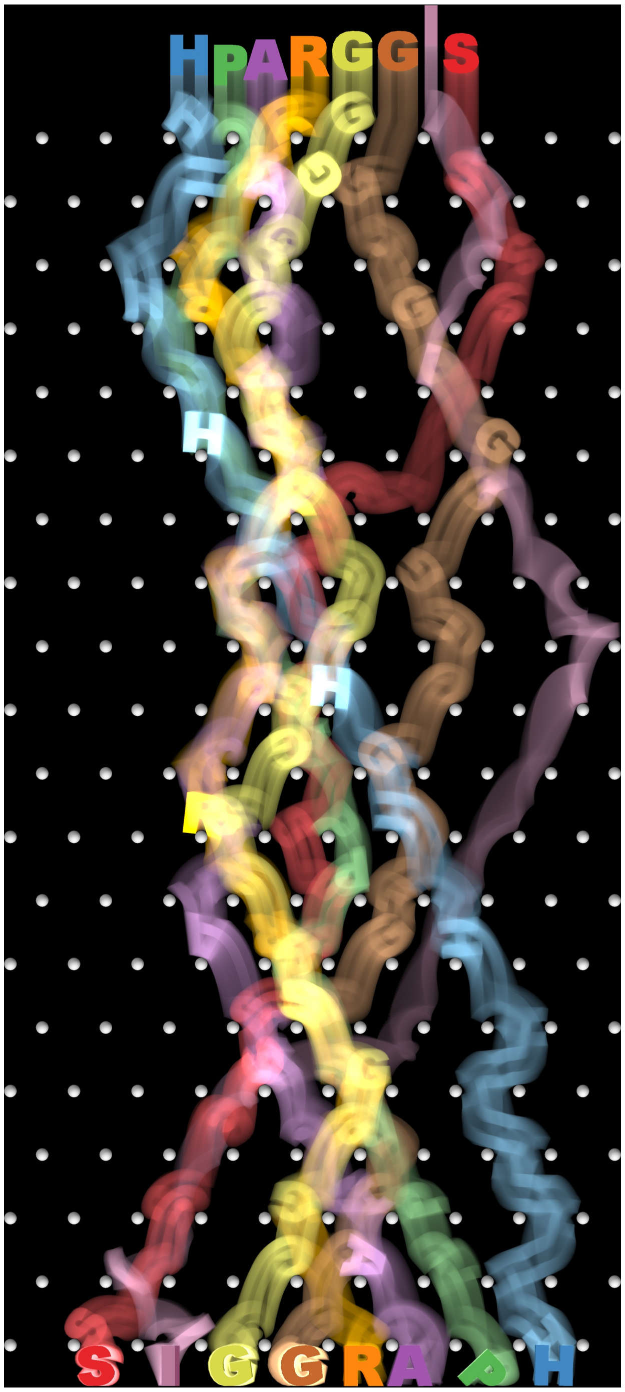


Figure 4.13: Browsing trajectories: *(left) In our* **refrigerator toy** *example, rotating buckets can be added even after sampling has begun; (right) after precomputation, over 1500 simulations with this* **trebuchet and tower** *can be loaded and browsed interactively.*

in the desired direction. Thanks to the visualization of paths, it is easy for the user to notice in a particular configuration if the ball is never going to land in the cup and quickly adjust the cup's position. Similarly, the use can carefully size the chutes such that the ball barely reaches the end without rolling off, increasing anticipation. The user is able to build a particular chain of four rotating cups in just under five and a half minutes.

While this is the simplest of the examples presented here, in many ways it seems to match most closely to the types of simulations that make up the bread and butter of production work. Whereas the other examples have somewhat arbitrary constraints imposed from the start, the refrigerator toy is built up in an organic fashion: the user uses the previously-computed simulation to make an aesthetic judgment about what should be done next. This style of working has the potential for broader adoption compared to more traditional optimization approaches; whereas a method such as Chenney and Forsyth [CF00] is clearly intended to handle special cases, there is no reason that a building-and-browsing approach similar to the one used to develop the refrigerator toy could not be made the default mode of interaction with the simulator. We could imagine the simulation getting carried all the way from pre-visualization through production in this fashion, as the scene is fleshed out piece by piece with improved geometry and other components.

Key to such an approach is the repeatability inherent in the Many-Worlds Browsing browser. Whereas in traditional simulations even adding an object far removed from the previously computed scene could affect dynamics: as the change is propagated through collision hierarchies and other structures, tiny differences are easily introduced and magnified by the chaotic nature of simulation. By storing the motion and recomputing exactly what is necessary, we substantially improve the stability of the simulation from the user's perspective, leaving him free to focus on the parts that need to be changed rather than trying vainly to recapture motion that has been irrevocably lost due to small adjustments in the scene.



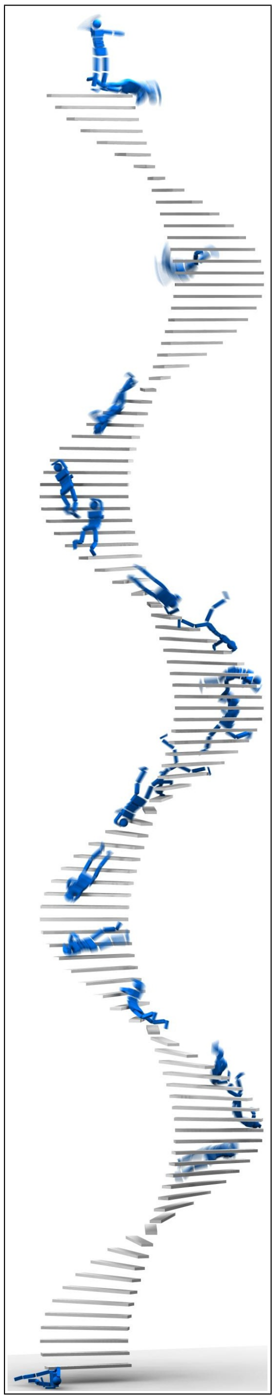


Figure 4.14: Spelling SIGGRAPH: *Using our parallel refinement with spatial queries and metrics enables a user to generate this animation spelling out "SIGGRAPH" from an arbitrarily chosen starting configuration.* **Spiral staircase:** *Using Many-Worlds Browsing, we can "guide" the character all the way down this spiral staircase.*

4.6.2 Spelling SIGGRAPH

The large number of collisions in this Pachinko-style environment (see Figure 4.14, left) means that we see a great deal of variation in each particular letter's individual motion. Nonetheless, getting all eight letters to land in their particular spots would be very difficult to do with simple random sampling–a simple estimate based on optimistically independent left/right Bernoulli trials suggests a probability on the order of 10^{-11}. Empowered with the ability to refine individual paths as independently as possible (subject to physical constraints), a user can guide the simulation toward the desired solution. This example took a few seconds more than an hour to generate; the user was forced to backtrack several times when it proved impossible to get the last few letters in without disturbing the rest.

In this case, the key aspect was the user's ability to quickly develop useful strategies for guiding the optimization. Utilizing the user's own "wetware" to solve problems that remain out of reach for offline optimizer is reminiscent of the approach used by Laszlo and colleagues [LvdPF00]. in our case, the primary discoveries included:

1. Due to the many potential collision points, it was relatively easy to fix up the orientation using the last few collisions at the end of the simulation. This allowed the user to focus on getting the letters into their "bins" and correct orientation in a second pass (important as we do not currently have a good method for visualizing orientation).

2. Initial attempts to get the letters into bins one by one proved frustrating as it proved difficult to get the last letter into place; that is, if we start with the letters in the reverse order "HPARGGIS" and land the letters "HPARGGI" into their alloted places, we tend to find that the letter S tends to interact with a substantial number of the previously-simulated letters as it crosses from one side to the other, thereby undoing the careful effort that was made ensuring that the letters land in their appointed bins. To address this it was necessary to take a more holistic view of the letters: the user first guided each of the letters into the correct general neighborhoods without worrying about exact placement, and then used a second pass to guide the objects into the correct bins.

3. Because our system does not handle orientations well, we used a simple metric measuring the angle difference from the object's starting orientation to find simulations in which the appropriate letter landed upright. Combining this with a basic position constraint was sufficient to find simulations where the letter landed in the correct bin and with the correct orientation.

It would be possible to develop algorithms that used each of these individual insights, but hard to develop an algorithm that could generalize as quickly as a person can. Nonetheless, it seems as though we could use these insights to provide more sophisticated search primitives like "get each of these objects in the correct neighborhood."

4.6.3 Spiral staircase

The goal here was to generate an animation where a simple articulated character model falls all the way down a three-level spiral staircase without falling off (see Figure Figure 4.14, right). Finding a single sample where the character falls all the way down the staircase would be extremely unlikely, but by using refinement we can generate improbable sequences of events by seeking out rare events and chaining them. Here the user is also able to cull out feasible yet uninteresting animations where the character simply slides down the staircase on his tail bone. For this animation, we applied a simple set-point controller to the character causing him to try to cover his head. This example took 25 minutes to generate, during which the user applied the refinement operator 18 times.

4.6.4 Trebuchet and tower

Our approach can be easily scaled up to larger models, but some models may require offline computation. We demonstrate this on a model consisting of a tower built from 192 stacked blocks being demolished by a functioning trebuchet mechanism. Due to ODE's relatively slow performance handling large object stacks, precomputing the 1549 samples in the dataset took 14 computing hours in total (which were spread across a cluster). Using a much faster solver, or even better hardware-accelerated physics, would eliminate this problem. Once the examples are computed, however, the user can easily browse the data set interactively using our system (see Figure 4.13). By randomizing various trebuchet parameters such as the release time for the thrown rock and the orientation of the base we reduced the amount of parameter fine-tuning required to produce useful results. Due to the high cost of computing for this data set, we were unable to perform refinement in real time, so our interactions with it were limited to making exploratory queries such as: "give me all examples where the rock hits the tower" or "... where the falling blocks miss the bystanders."

Chapter 5

Backward simulation

5.1 Introduction

There are a multitude of schemes for time-stepping rigid body simulations. While it may be possible to turn many of these into backward schemes, there are various trade-offs. In particular, simple penalty-based methods can afford at best a minimal amount of control over the resulting simulation, as small oscillations in position due to numerical round-off quickly grow into large bounces due to the use of inverse damping, which makes capturing either sliding or stacking behavior difficult. Therefore, we based our scheme on Linear Complementarity Problem formulations due to Baraff [Bar91] and Stewart and Trinkle [ST97] that were discussed in Chapter 3.

Terminology: Before discussing time-reversed simulation in depth, we should pause to establish some terminology. We will use the term *forward* to refer to *increasing* time: $t, t + \Delta t, t + 2\Delta t, \ldots$. *Backward* and *reverse* will refer to decreasing time: $t, t - \Delta t, t - 2\Delta t, \ldots$. Similarly, with respect to a reference time t_{ref}, *before* will refer to times $t < t_{ref}$ and *after* will refer to times $t > t_{ref}$. Although we will refer to forward simulation frequently, the problems we will actually be solving will all use backward integration, so the *initial conditions* for the solver will always refer to the simulation state we provide as input to the reverse solver, which will often (but not always) consist of the bodies sitting at rest.

5.2 Backward steps

The complete state for a rigid body j at time t consists of its position $\mathbf{x}_j^t$, its orientation $\mathbf{\Theta}_j^t$, its linear velocity $\mathbf{v}_j^t$, and its angular velocity ω_j^t. In backward simulation, we are given the complete system state $\{\mathbf{x}_j^t, \mathbf{\Theta}_j^t, \mathbf{v}_j^t, \omega_j^t\}$ at time t and want to compute the state $\{\mathbf{x}_j^{t-\Delta t}, \mathbf{\Theta}_j^{t-\Delta t}, \mathbf{v}_j^{t-\Delta t}, \omega_j^{t-\Delta t}\}$ at time $t - \Delta t$. Recall that we use *"initial conditions"* to refer to the starting conditions of the backward simulation: the state at time t.

We will consider a state $\{\mathbf{x}_j^{t-\Delta t}, \mathbf{\Theta}_j^{t-\Delta t}, \mathbf{v}_j^{t-\Delta t}, \omega_j^{t-\Delta t}\}$ to be plausible if the *forward* timestepping

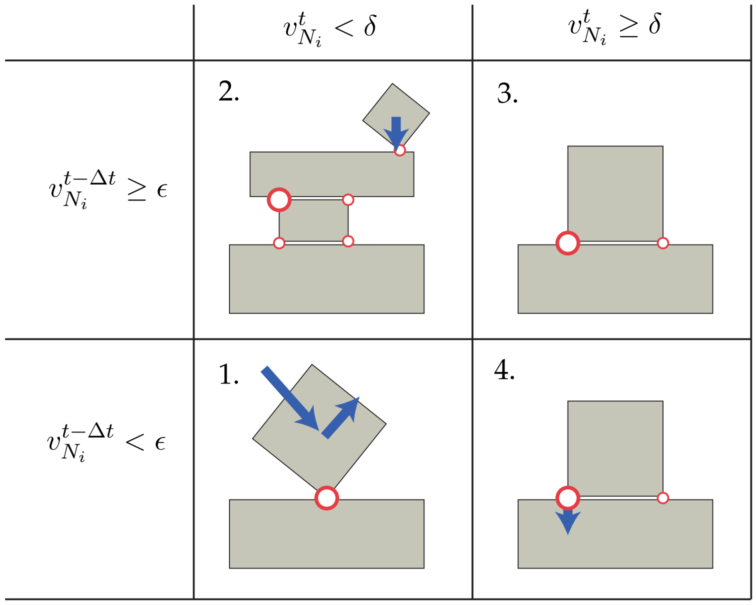

$v_{N_i}^t < \delta$
$v_{N_i}^t \geq \delta$
$v_{N_i}^{t-\Delta t} \geq \epsilon$
$v_{N_i}^{t-\Delta t} < \epsilon$
2.
3.
1.
4.

Figure 5.1: Four possibilities for contact behavior during forward simulation: *We can break down the possibilities along two axes based on whether we see a non-negligible pre-timestep velocity $v_{N_i}^{t-\Delta t}$ and/or a significant post-timestep velocity $v_{N_i}^t$. Clockwise from bottom left, we have* **(1)** *the object's incoming velocity was such that the simulator applied an elastic collision,* **(2)** *the object was sitting at rest but due to other forces/contacts in the system the object's velocity is nonzero at the end of the timestep.* **(3)** *the object is sitting at rest at both frames, and* **(4)** *the object had a small enough incoming velocity that the simulator decided to apply an inelastic collision.*

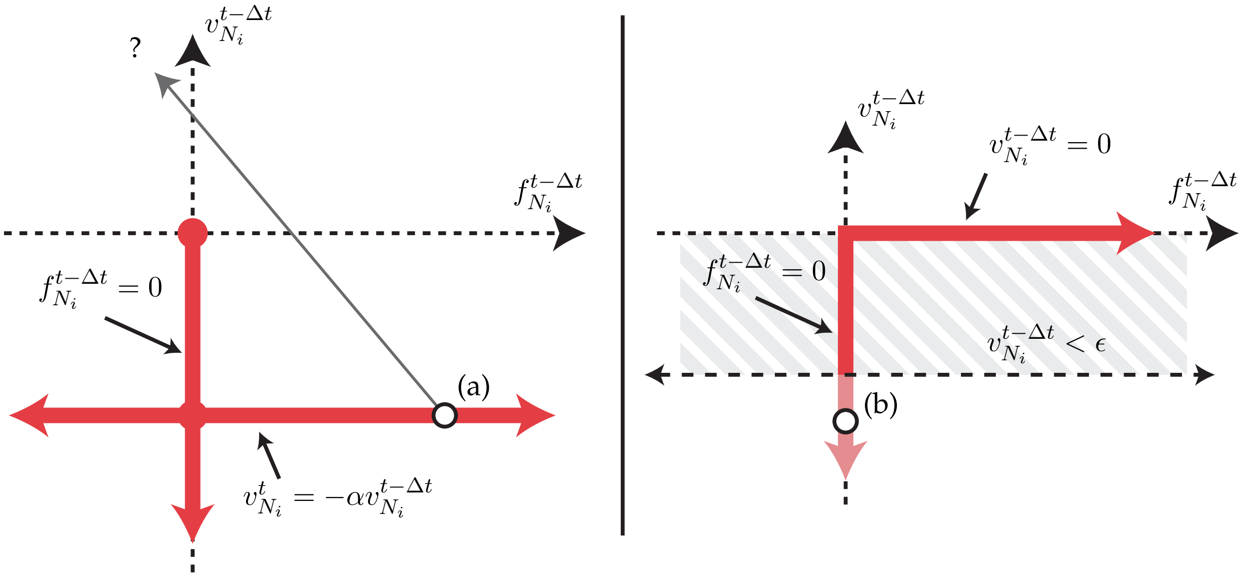
scheme described in Chapter 3 would if given $\{\mathbf{x}_j^{t-\Delta t}, \mathbf{\Theta}_j^{t-\Delta t}, \mathbf{v}_j^{t-\Delta t}, \omega_j^{t-\Delta t}\}$ as input produce the $\{\mathbf{x}_j^t, \mathbf{\Theta}_j^t, \mathbf{v}_j^t, \omega_j^t\}$ as output. We will examine each kind of constraint separately and break down the possible configurations at $t - \Delta t$.

5.2.1 Normal forces

Consider first a normal constraint i (Figure 3.1, left). We can easily compute the relative normal velocity $v_{N_i}^t$ of the two bodies at this point. We know that $v_{N_i}^t \geq 0$ (to machine precision) as this is a simulation invariant, and to prevent interpenetration we will require that $v_{N_i}^{t-\Delta t} \leq 0$. Based on the LCP formulation, there are four distinct possibilities for what could have happened at time $t - \Delta t$ such that forward simulation would have brought us to our current state (see also Figure 5.1),

1. The relative velocity $v_{N_i}^{t-\Delta t}$ was large enough that the solver chose to apply an elastic collision response; that is, $v_{N_i}^t \geq -\alpha v_{N_i}^{t-\Delta t}$.

2. The relative velocity $v_{N_i}^{t-\Delta t}$ was 0 at $t - \Delta t$ but is positive at the current time; $v_{N_i}^{t-\Delta t} = 0$ and $v_{N_i}^t > 0$. In this case, we can deduce that $a_{N_i}^{t-\Delta t} > 0$, which implies that $f_{N_i}^{t-\Delta t} = 0$ (from the LCP conditions).



$v_{N_i}^{t-\Delta t}$
?
$f_{N_i}^{t-\Delta t}$
$f_{N_i}^{t-\Delta t} = 0$
(a)
$v_{N_i}^{t} = -\alpha v_{N_i}^{t-\Delta t}$
$v_{N_i}^{t-\Delta t}$
$v_{N_i}^{t-\Delta t} = 0$
$f_{N_i}^{t-\Delta t}$
$f_{N_i}^{t-\Delta t} = 0$
$v_{N_i}^{t-\Delta t} < \epsilon$
(b)

Figure 5.2: LCP conditions when simulating backwards: Left: *If we find ourselves at time t with a strictly positive $v_{N_i}^t$, there are two possibilities for what could have happened at time $t - \Delta t$: either we applied an elastic collision response (corresponding to the horizontal line here) or we applied an inelastic response but nonetheless saw a nonzero post-timestep velocity $v_{N_i}^t$ (corresponding to the vertical line here). Given the other forces in the system, it is trivial to determine whether a nonzero $f_{N_i}^{t-\Delta t}$ is needed to explain why $v_{N_i}^t > 0$; with multiple constraints, however, it is unclear how we can quickly determine* which *of the $f_{N_i}^{t-\Delta t}$ lay on the vertical line and which lay on the horizontal line. (Note: Point (a) is discussed in the text.)* **Right:** *Under the restitution model we use during forward simulation, the only way that $v_{N_i}^t$ can be zero is if $|v_{N_i}^{t-\Delta t}| < \epsilon$. However, if we add this constraint the LCP solver will not converge in many cases, as if we are at point (b) we are not allowed to drive $f_{N_i}^{t-\Delta t}$ into the negative region.*

3. There was no relative motion normal to the surfaces at either $t - \Delta t$ or t; that is, $v_{N_i}^{t-\Delta t} = v_{N_i}^t = 0$.

4. The relative velocity was strictly negative at $t - \Delta t$ but is 0 at the current time; $v_{N_i}^{t-\Delta t} < 0$ and $v_{N_i}^t = 0$. While $a_{N_i}^{t-\Delta t} > 0$, this does *not* necessarily imply that $f_{N_i}^{t-\Delta t} = 0$, as the LCP conditions allow the acceleration necessary to cancel an interpenetrating velocity. We can, however, infer that the prior velocity $v_{N_i}^{t-\Delta t}$ was small enough in magnitude to not activate an elastic response.

At run time, we can easily distinguish between the two cases with $v_{N_i}^t > 0$ (1. and 2.) and the two cases with $v_{N_i}^t = 0$ (3. and 4.), as $v_{N_i}^t$ is given as input. Thus, we break our contact handling into two paths, using a small cutoff velocity $\delta \ll \epsilon$ to distinguish between the cases,

$$v_{N_i}^t < \delta \qquad\qquad v_{N_i}^t \geq \delta \tag{5.1}$$

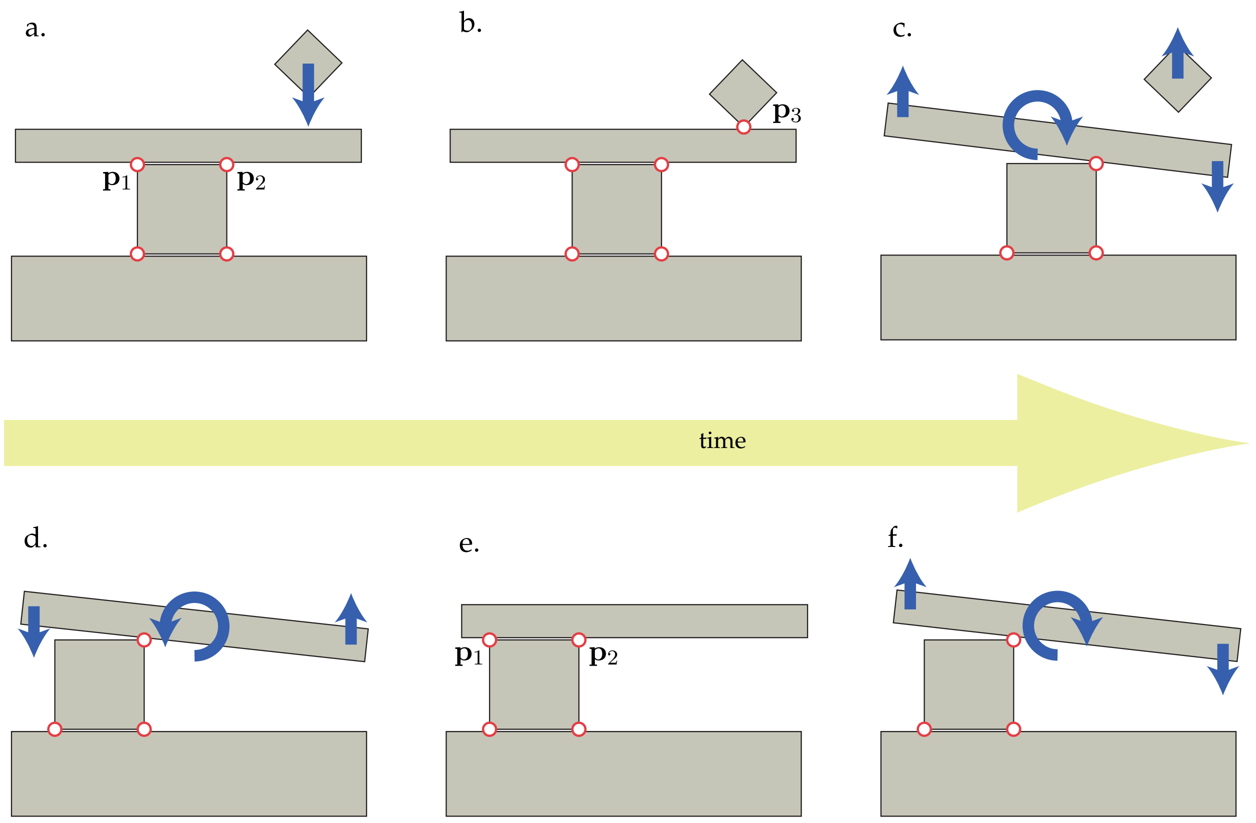


Figure 5.3: Explaining a nonzero $v^t_{N_i}$ without elastic impulses. *If we look at the point $\mathbf{p}_1$ in these two examples, we notice that v_{N_1} is nonzero in frames* **(c)** *and* **(f)** *without any elastic impulses being applied at $\mathbf{p}_1$. In the top example* **(a-c)**, *this occurs due to an elastic impulse applied at $\mathbf{p}_3$. In the bottom example, however, no elastic impulses are needed and the resulting motion is merely the result of the initial state being unstable (it would not be surprising if an animator were to provide frame* **(e)** *as an initial state for a simulation). Reaching state* **(e)** *via backward simulation from* **(f)** *or via forward simulation from* **(d)** *would require an extraordinarily fortuitous combination of velocities and forces, which is generally the case for these sorts of events.*

5.2.2 LCP normal conditions when $v^t_{N_i} \geq \delta$

Overall, the two cases with $v^t_{N_i} > 0$ are the most difficult to handle: essentially, we are required to decide whether an object's outgoing post-collision velocity resulted from (i) an elastic collision applied at that contact, or (ii) other forces in the system. This ambiguity is a stumbling block for the LCP solver; if we consider Figure 5.2 we can see the two possible lines that $\{f^{t-\Delta t}_{N_i}, v^{t-\Delta t}_{N_i}\}$ are allowed to lie on. Barring a fortuitous combination of body forces, the only way we will ever see both $v^{t-\Delta t}_{N_i} > 0$ and $f^{t-\Delta t}_{N_i} = 0$ is if some other $f^{t-\Delta t}_{N_j}$ is nonzero ($j \neq i$), as can be seen for example in Figure 5.3. If we want to find cases where $f_{N_i} = 0$ and $v^{t-\Delta t}_{N_i}$ we will have to experiment with setting other f_{N_i} in the system to nonzero values, which turns our linear LCP formulation into a combinatorial optimization problem. From another perspective, imagine we are at point (a) in Figure 5.2 and the LCP solver needs to decide what to do. If we try to push $f^{t-\Delta t}_{N_i}$ toward 0, there is a significant risk that the slope of the line will be too steep and we will miss the valid region altogether (it is impossible to know the slope exactly *a priori*, as changes to the index sets will affect it). So the safe course for the solver is to simply keep $v^{t-\Delta t}_{N_i}$ on the horizontal line, which it can always do by applying appropriate forces. In an LCP solver using Lemke's algorithm which processes the variables one by one, variables which reach the horizontal line can never leave it, so the order in which we handle variables will change the result we get.

For this reason, we discard possibility 2 altogether; if we see that $v^t_{N_i} \geq 0$, we assume that an elastic collision was applied, which under the Newton restitution model means that

$$v^t_{N_i} = -\alpha v^{t-\Delta t}_{N_i}. \tag{5.2}$$

As we cannot guarantee that in an arbitrary frictional system a solution will exist in which each of the contacts satisfies the equality, we relax (5.2) by posing it as a linear complementarity problem,

$$0 \leq v^{t-\Delta t}_{N_i} + v^t_{N_i}/\alpha \qquad \perp \qquad f^{t-\Delta t}_{N_i} \geq 0 \tag{5.3}$$

This can produce somewhat different results from forward simulation; whereas in forward simulation we guarantee that the post-timestep velocity $v^t_{N_i}$ is *at least* as large as the desired elastic response given the pre-timestep velocity $v^{t-\Delta t}_{N_i}$, in our backward simulator the velocity $v^t_{N_i}$ is *at most* the elastic response.

5.2.3 LCP normal conditions when $v^t_{N_i} < \delta$

For any contact i where $v^t_{N_i} \approx 0$, we know as discussed above that a forward solver could not have processed this as an elastic collision because our elastic collision model guarantees a nonzero post-collision velocity. We can therefore assume that an inelastic collision or persistent contact occurred at time $t - \Delta t$. Thus, we have the constraint,

$$-v^{t-\Delta t}_{N_i} < \epsilon. \tag{5.4}$$

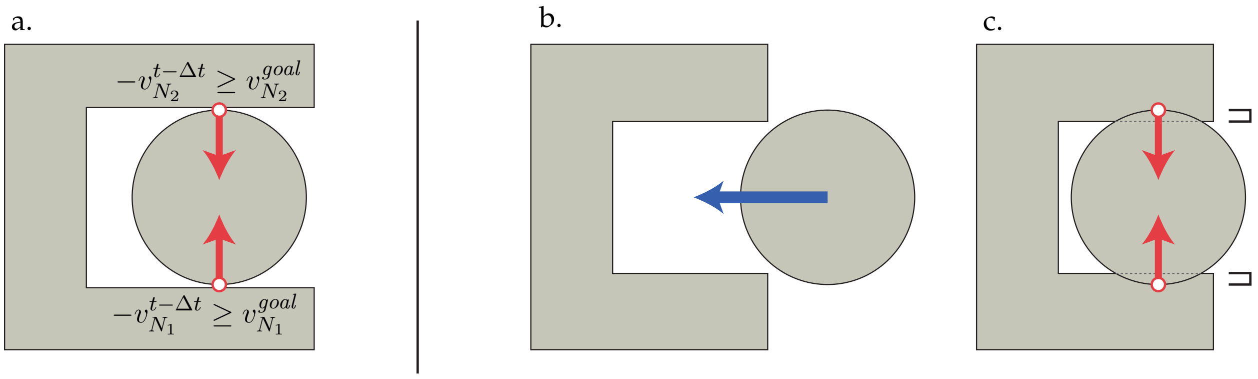

a.
$-v_{N_2}^{t-\Delta t} \geq v_{N_2}^{goal}$
$-v_{N_1}^{t-\Delta t} \geq v_{N_1}^{goal}$
b.
c.

Figure 5.4: Conflicting constraints: (a) *If we require that the incoming velocity at each of these contacts be nonzero, no solution can possibly exist. We must either prevent the user from providing conflicting constraints like this (which may be difficult as other cases may be more subtle) or rely on our constraint solver to detect these cases and return a reasonable result. Conflicting constraints are not just limited to backward simulation, however; in* **(b-c)** *in the absence of continuous collision detection, an overly large forward timestep allows this sphere to step too far into this cup; here, collision normals are detected perfectly, but no amount of force applied along these normals can correct the interpenetration (as Stewart's and Trinkle's LCP formulation requires). Of course, applying a rightward impulse would work, but the LCP solver is not allowed to change collision normals.*

Unfortunately, it is difficult to guarantee that (5.4) will hold due to the influence of the other $f_{N_j}^{t-\Delta t}$ for $j \neq i$, as these may push $v_{N_i}^{t-\Delta t}$ over the ϵ threshold, and we are not allowed to apply a negative $f_{N_i}^{t-\Delta t}$ (see Figure 5.2).

In addition, we want to address the ambiguity inherent in resting contact by giving the user control: the user should be able to specify whether an object was sitting at rest at time $t - \Delta t$ (subject to other forces in the system, of course) or whether the object had a nonzero velocity $v_{N_i}^{t-\Delta t}$ that was damped out for being below the cutoff velocity ϵ. If we do not provide this choice, our objects at rest will forever remain objects at rest and the simulations that result will be rather dull. We therefore allow the user to choose a goal velocity $v_{N_i}^{goal}$, where setting $v_{N_i}^{goal} = 0$ means that (subject to other constraints) the relative velocity between the objects will remain 0, while $0 < v_{N_i}^{goal} < \epsilon$ instructs the solver to provide a damped collision response at time $t - \Delta t$, $-v_{N_i}^{t-\Delta t} \geq v_{N_i}^{goal}$. We incorporate this into the solver through an LCP condition,

$$0 \geq v_{N_i}^{t-\Delta t} + v_{N_i}^{goal} \qquad \perp \qquad f_{N_i}^{t-\Delta t} \geq 0 \tag{5.5}$$

The result of (5.5) is to require that $-v_{N_i}^{t-\Delta t}$ be at least $v_{N_i}^{goal}$ and that if $-v_{N_i}^{t-\Delta t} > v_{N_i}^{goal}$, it is not due to force $f_{N_i}^{t-\Delta t}$ applied at point i.

When applying constraints like (5.5), there is a risk that we will generate an LCP system that has no feasible solution (see Figure 5.4). The correct thing to do in this case is to de-prioritize the $v_{N_i}^{goal}$ constraints as they are really just suggestions. We do not perform any explicit fall-back in this case but instead rely on our constraint solver to do something reasonable, which in cases like

Figure 5.4 is likely to involve some interpenetration at either N_1 or N_2. It would not be hard, however, to implement a system whereby the constraints $v_{N_i}^{t-\Delta t} \leq 0$ are treated first by the solver, and constraints $v_{N_i}^{t-\Delta t} \leq -v_{N_i}^{goal}$ are considered later but only if they do not cause violations of earlier constraints.

5.2.4 LCP friction conditions

In the frictional LCP for forward simulation described in §3.6, the friction force was required to maximally oppose the tangential velocity at a point if that velocity was nonzero. Under this model, if $v_{\mathbf{w}_i}^t > 0$, then friction must have been maximal in the previous timestep and opposing $v_{\mathbf{w}_i}^t$. This is the condition,

$$|f_{\mathbf{w}_i}| = \mu f_{N_i}^{t-\Delta t} \qquad\qquad (f_{\mathbf{w}_i})(v_{\mathbf{w}_i}^t) \leq 0 \tag{5.6}$$

On the other hand, if $v_{\mathbf{w}_i}^t = 0$, then $f_{\mathbf{w}_i}^{t-\Delta t}$ could lie anywhere inside the friction cone; this is underconstrained, as we would expect it to be, because damping forces tend to remove information from the system. While any answer returned by the solver is guaranteed to be feasible, the user can "request" a particular $v_{\mathbf{w}_i}^{t-\Delta t}$ by providing a goal velocity $v_{\mathbf{w}_i}^{goal}$; the resulting LCP becomes

$$0 \leq \mu f_{N_i}^{t-\Delta t} - |f_{\mathbf{w}_i}| \qquad \perp \qquad v_{\mathbf{w}_i}^{t-\Delta t} - v_{\mathbf{w}_i}^{goal} \tag{5.7}$$

Under this model, the friction force will act to drive the tangential velocity force toward $v_{\mathbf{w}_i}^{goal}$, but is not allowed to exceed the friction cone conditions.

We note that physics does not require us to specify that friction force oppose velocity; the original principle of maximal dissipation for forward simulation stated that the post-timestep velocity $v_{\mathbf{w}_i}^t$ be the minimum one out of all the possible velocities satisfying the friction cone constraints, and certainly when $v_{\mathbf{w}_i}^t = 0$ this is trivially true. However, if we do not add the constraint

$$(f_{\mathbf{w}_i}^{t-\Delta t})(v_{\mathbf{w}_i}^{t-\Delta t} - v_{\mathbf{w}_i}^{goal}) \leq 0 \tag{5.8}$$

then $|f_{\mathbf{w}_i}| = \mu f_{N_i}^{t-\Delta t}$ would be considered a valid solution, even if $f_{\mathbf{w}_i}$ is driving $v_{\mathbf{w}_i}^{t-\Delta t}$ *away* from the desired goal velocity.

We can unify the treatment of (5.6) and (5.7) by letting $v_{\mathbf{w}_i}^{goal}$ in (5.6) go to $+\infty$ or $-\infty$, depending on the sign of $v_{\mathbf{w}_i}^t$. Then, as $v_{\mathbf{w}_i}^{t-\Delta t} - v_{\mathbf{w}_i}^{goal} \neq 0$, the left-hand side of (5.7) becomes equality, and by selecting between $\pm\infty$ for $v_{\mathbf{w}_i}^{goal}$ we can ensure that $v_{\mathbf{w}_i}^{t-\Delta t} - v_{\mathbf{w}_i}^{goal}$ in (5.8) and $v_{\mathbf{w}}^t$ in (5.6) have the same sign.

For reasons described in §3.6.1, we get better results if one of our two friction directions lines up with the existing tangential velocity direction. If the tangential velocity at time t is 0, we can instead choose $\mathbf{w}_i$ to line up with the user's chosen direction.

5.2.5 Implementation

Our timestepping scheme for backward simulation is just a mirror of (3.15)-(3.18). After computing the forces $\mathbf{f}_j^{t-\Delta t}$ and torques $\tau_j^{t-\Delta t}$, we advance the velocities,

$$\mathbf{v}_i^{t-\Delta t} = \mathbf{v}_i^t - \Delta t \mathbf{a}_i \tag{5.9}$$
$$\omega_i^{t-\Delta t} = \omega_i^t - \Delta t \alpha_i \tag{5.10}$$

which are then used to advance the positions in the normal way.

As before, the primary focus of the solver is on computing the forces and torques that will be used to advance the linear and rotational velocities. Solving for contact forces requires only a working LCP solver. Our system uses ODE's implementation of Lemke's algorithm, which we have modified to more tightly couple the friction and normal forces. The number of constraints in the system is identical to those used in forward simulation, so performance is similar. The run-time performance of Lemke's algorithm is dominated by the cost of repeated matrix solves, and if fast $O(n^2)$ matrix updates are used the total cost is roughly cubic in practice. However, like the simplex method on which it is based, Lemke's algorithm has exponential worst-case performance [Bar94].

The linear relationship (3.13) must be changed due to the backward timestepping scheme,

$$\mathbf{v}^{t-\Delta t} = \mathbf{v}^t - \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{f} - \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_0 \tag{5.11}$$

Since most LCP solvers are designed to operate in terms of positive definite rather than negative definite matrices, we can flip signs,

$$-\mathbf{v}^{t-\Delta t} = -\mathbf{v}^t + \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{f} + \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_0 \tag{5.12}$$

Fortunately, our constraints are most naturally posed in terms of negative velocities, so no changes are needed to the solver to support (5.12). Now, we observe that the constraints for backward simulation all include a "goal velocity" term $v_{\mathbf{w}_i}^{goal}$ or $v_{N_i}^{goal}$. These can be easily added to (5.12),

$$-(\mathbf{v}^{t-\Delta t} + \mathbf{v}^{goal}) = -\mathbf{v}^t + \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{J}^T\mathbf{f} + \Delta t \mathbf{J}\mathbf{M}^{-1}\mathbf{f}_0 \tag{5.13}$$

Selecting these goal velocities will depend on the criteria we discussed in the previous section. We can summarize them using the algorithm in Figure 5.5. It takes as input the relative velocity between the objects at the point and the contact normal $\mathbf{n}_i$.

5.3 Addressing Challenges and Limitations

While it is almost always *possible* in a local sense to step a given simulation in the backward direction, naïvely applying to a given initial condition can generate motion that is both qualitatively and quantitatively different from motion generated by a forward simulation. It is important to note that this is *not* a numerical artifact of the solver; that is, even if we could guarantee that in

```
HANDLE-CONTACT($\mathbf{v}_i^{t-\Delta t}, \mathbf{n}_i$)
 1  $v_{N_i}^t \leftarrow \mathbf{v}_i^t \cdot \mathbf{n}_i$    *Splitting velocity into components*
 2  $\mathbf{w}_i \leftarrow (\mathbf{v}_i^{t-\Delta t} - v_{N_i}^t \mathbf{n}_i) / \left|\left|\mathbf{v}_i^{t-\Delta t} - v_{N_i}^t \mathbf{n}_i\right|\right|$
 3  $v_{\mathbf{w}_i}^t = \mathbf{v}_i^t \cdot \mathbf{w}_i$
 4  **if** $v_{\mathbf{w}_i}^t < \delta$          *Setting user's friction direction*
 5     **then** $\mathbf{w}_i \leftarrow$ user preference
 6  **switch**              *Normal velocity needs bounce?*
 7     **case** $v_{N_i}^t > \delta$ :
 8           $v_{N_i}^{goal} \leftarrow -v_{N_i}^t / \alpha$
 9     **case** $v_{N_i}^t \leq \delta$ :
10           $v_{N_i}^{goal} \leftarrow 0$ or user preference
11  **switch**              *Friction force needs to be maximal?*
12     **case** $v_{\mathbf{w}_i}^t \leq \delta$ :
13           $\mathbf{w}_i \leftarrow$ user-preferred direction
14           $v_{\mathbf{w}_i}^{goal} \leftarrow$ user-preferred speed
15     **case** $v_{\mathbf{w}_i}^t > \delta$ :
16           $v_{\mathbf{w}_i}^{goal} \leftarrow \infty$
```

Figure 5.5: Algorithm for handling contacts during backward simulation: *As described in §5.2.5, virtually all the special-case code for backward simulation is in the contact handling, where the goal normal and tangential velocities are set. The relative velocity at the point is broken into the two components, at which point we must take into account the current velocities and user preferences in setting $v_{\mathbf{w}_i}^{goal}$ and $v_{N_i}^{goal}$.*

all situations taking a backward step followed by a forward step (or vice versa) would bring us back exactly to where we started, our backward simulations will still be highly sensitive to the problems we list here. In most cases, after all, we are not providing the simulator with starting conditions that are the result of any particular forward simulation; rather, the initial conditions for the backwards solver come from our own imaginations. In many cases we may provide starting conditions that are not the result of *any* valid forward simulation, and these innocent-seeming but invalid initial conditions are the cause of the problems we describe here.

We will illustrate these problems with a combination of simple thought experiments and numerical results.

5.3.1 Rapid velocity growth:

Backward simulation can lead to rapid growth in simulation energy as a result of restitution-related collisional and frictional dissipation effects being run in reverse. In practice, simulations with collisions can quickly become excited, and effectively have useful time horizons which are short. One strategy for alleviating this problem, but not entirely removing it, is to make the simulation appear more reversible by using restitution coefficients closer to one, and by not having large friction coefficients.

Perhaps more troublesome than energy growth in general is the rapid growth of angular velocity specifically. When we run forward simulations, we tend to use initial conditions that have both linear and angular velocity set to zero. Linear velocity increases with gravity, and during collision events that energy is transferred to angular velocity. During reverse simulation, however, there is nothing driving angular velocities toward zero as $t \to -\infty$ and in fact they tend to grow without bound. While this is physically reasonable, it violates our expectations. Angular velocity growth is a particular problem for more oblong objects, as the forces applied during restitution are far from the center of mass, leading to large moments and high angular velocities.

Another possibility, although it is less physical, is to introduce a backward damping term to limit the rate at which simulations gain energy. We can apply a torque,

$$\tau_d = k_d \omega \tag{5.14}$$

Normally we would expect the sign in (5.14) to be flipped, but as torques get *subtracted* during a backwards timestep, formulating a damping force as in (5.14) acts to damp velocities during reverse simulation. Setting k_d too high causes the objects to appear to accelerate into contacts when the motion is played back, which is an unusual artifact. Small amounts of damping are harder to notice, but because energy grows exponentially as a function of the number of collisions and the reverse damping force (5.14) acts only as a function of time, collision-heavy scenes tend to render the damping term somewhat ineffectual. Heavily damping angular velocities also removes energy from the system, which causes the objects to “skittle” around during the played-back motion.

Unchecked micro-collision events can also lead to rapid energy gains for objects that can rattle between two other objects, such as a rubber ball trapped between panes of glass or the five balls

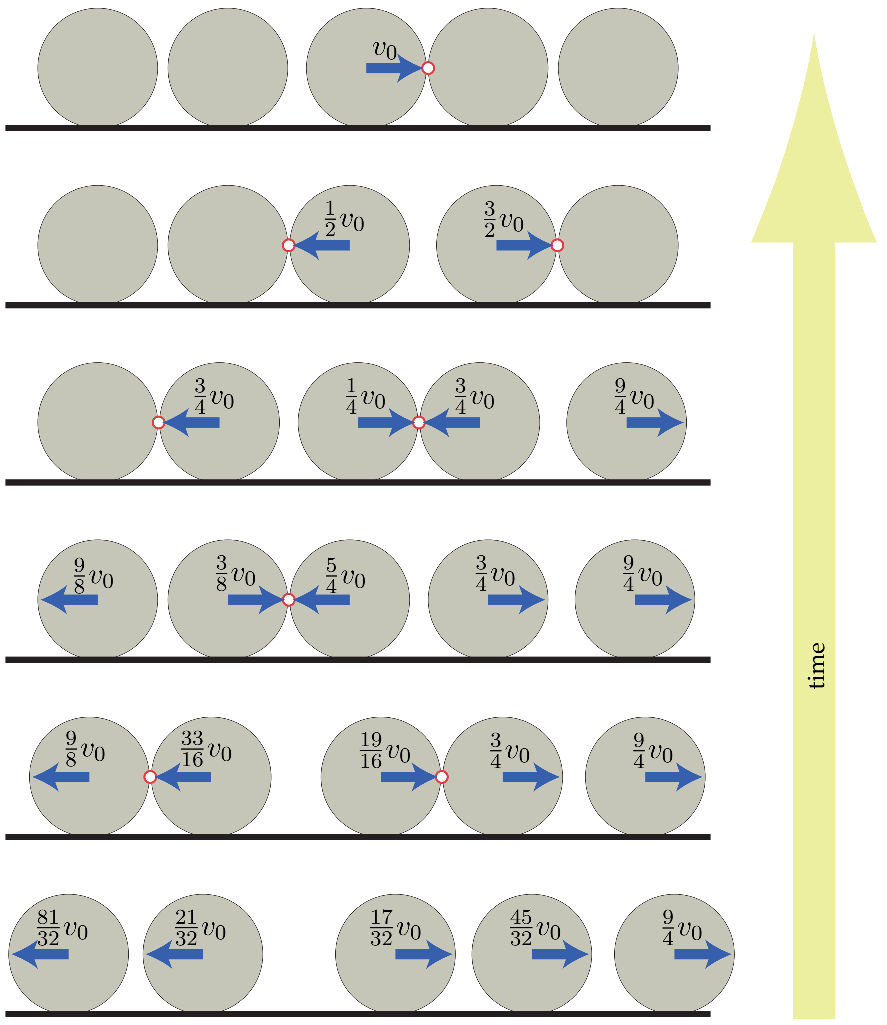


Figure 5.6: Rattling between objects: *If we have several objects close together, repeated collisions between objects can lead to rapid energy gain. Here the 5 balls are assumed to have identical masses and each pair has a restitution coefficient of $\frac{1}{2}$. Blue arrows indicate the reverse velocity directions; that is, the directions that our backward integrator moves the objects. If we give one object a nonzero initial velocity, after only a few backward steps the kinetic energy in the system increases by a factor of nearly 7. This problem is worse for smaller restitution coefficients and in 3D scenes due to repeated collisions between the same pair of objects (imaging a box rattling back and forth between two corners).*

in Figure 5.6. We can reduce the gain by increasing the restitution coefficient when we see many collisions between the same two objects in a short time period while running backward. In our system, the restitution coefficient between two objects is set to 1 whenever an elastic collision is processed between those objects and then smoothly blended back to the original value over a short time period (0.2 seconds). The visual effect of this change will be some additional rattling, causing the objects to take more time to come to rest.

5.3.2 Sensitivity to initial conditions

In a sense all the issues we describe could be grouped under this heading; however, we will use it specifically to refer to problems that are highly constrained. Consider the following simple experiment: we place a small box inside a larger box that has a single exit hole.

When we run this simulation backward, we find that the smaller box bounces around inside the larger one indefinitely while experiencing exponential energy growth due to the vanishingly small probability of finding the exit. While this is (mathematically) a valid simulation, it does not register to us as plausible because it seems unlikely that the smaller box could have started with such a large amount of kinetic energy. Certainly, low-probability events can be a problem for forward simulation as well, but the response (rapid energy growth) is qualitatively different.

Given that we know that there exist forward simulations that enter the box through the hole and settle to rest, the question becomes one of picking an initial condition for our backward simulation that when stepped backward will produce one of these forward simulations. In many cases, however, if we pick the initial conditions in a random or pseudo-random fashion we have an arbitrarily small probability of selecting an initial condition that is the result of some valid forward simulation. To see this, consider Figure 5.7, where it is evident that even a minimum of geometry can lead to a very nonlinear mapping from the input space to the output space.

5.3.3 Stacking

One example we would like to generate involves having objects bounce around and then coincidentally assemble into some kind of structure. To generate this, our simulator must be able to handle stacking behavior. Any kind of stacking immediately implies a causal direction to the simulation, as in forward simulation objects at the top of the pile cannot fall into place until objects at the bottom of the pile are there to fall on. The corresponding requirement when simulating backwards is that we only cause an at-rest object to slide, roll, or bounce if it is at the top of the pile. Although determining which objects should be free to bounce is a difficult problem in general, we use the simple heuristic of examining all the contact normals for a particular object; if the dot product of every pair of normals is positive, then the object is considered free to bounce, whereas if any pair of normals are opposing, the object is considered constrained. The user can override this heuristic if necessary using the sketching interface described later.

Even after we ensure causality through this heuristic, however, we note that stacks of bodies

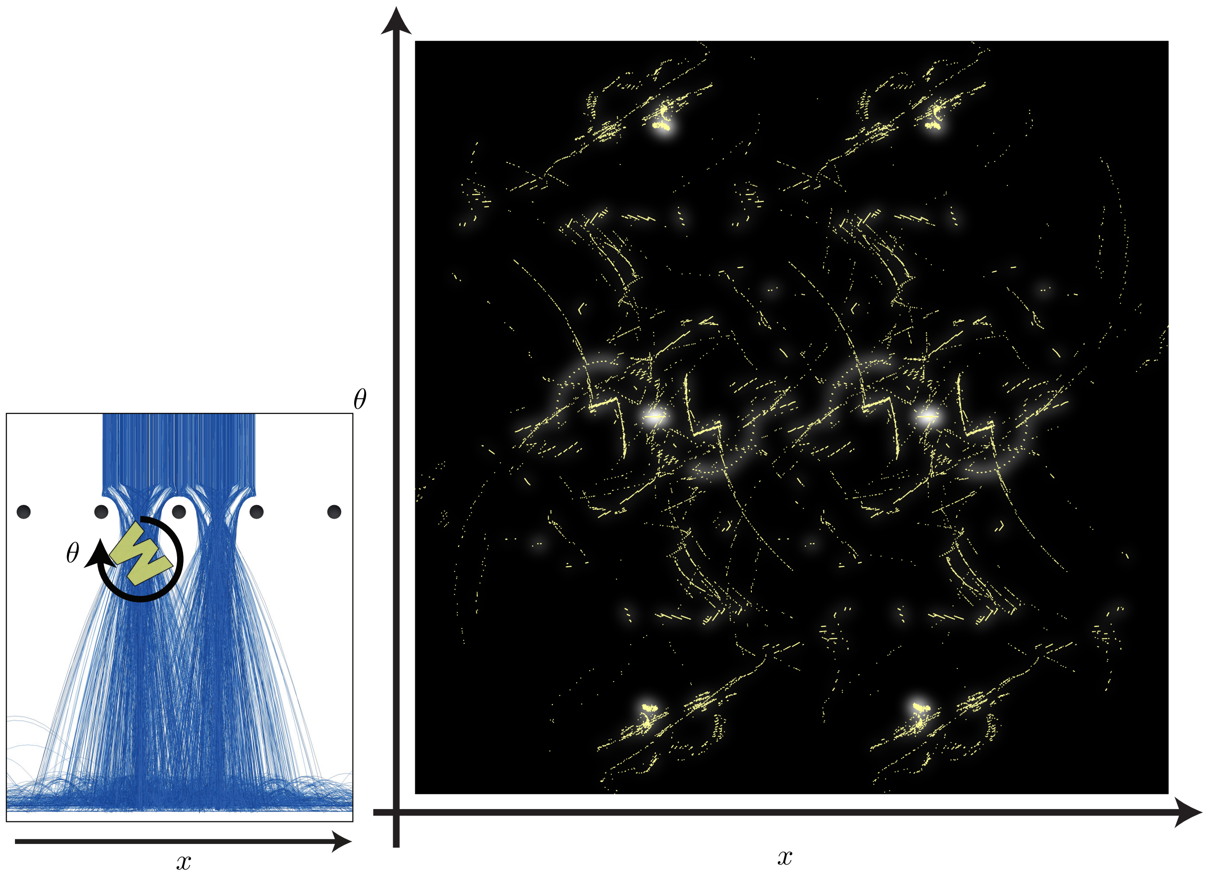


Figure 5.7: Sensitivity to initial conditions: **Left:** *If we uniformly sample 10,000 initial conditions we can compute the resulting motion and examine the final object configurations. To reduce quantization artifacts in the rotational component we actually look at the position and rotation when the object passes through a fixed height $y = h$.* **Right:** *These position/rotation pairs can then be plotted on a graph as shown (points are in yellow and overlaid on top of a density plot). The set of resulting configurations is thus a very thin manifold embedded in a higher dimensional space, and if we randomly sample an end configuration for starting our backward simulation, the odds that we will pick a point on that manifold are vanishingly small. This example gives some intuition as to why it will be difficult in heavily constrained problems to find resting configurations for starting our backward simulation that will produce plausible-looking motion with acceptable initial conditions.*

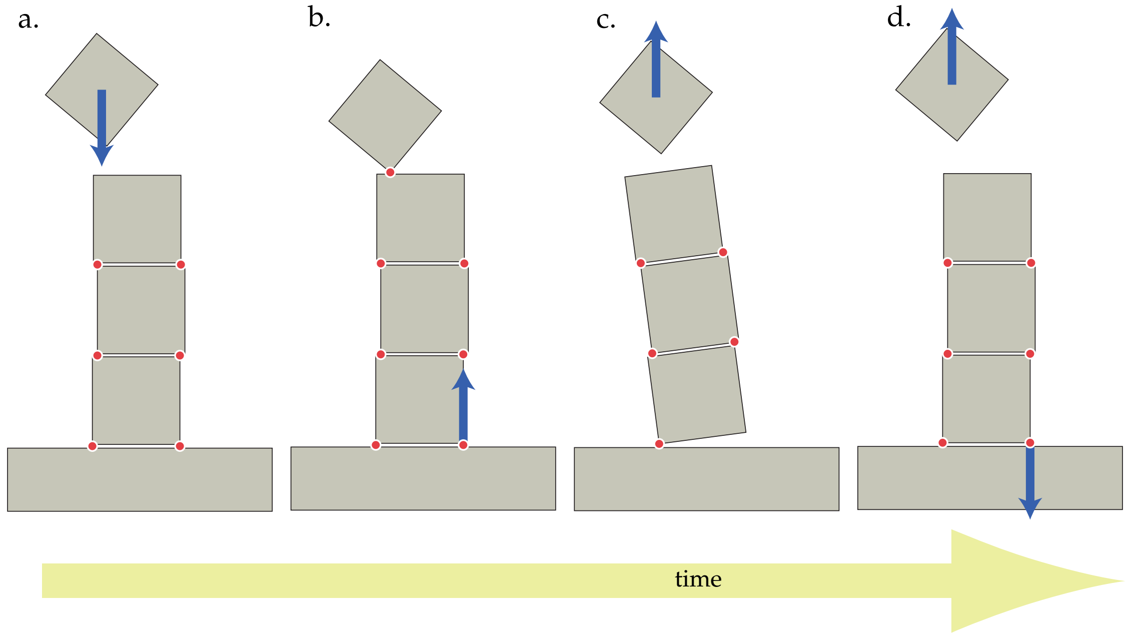


Figure 5.8: Issues with stacking behavior: *In* forward *simulation, a high-speed elastic collision between a single block and a stack of blocks* **(a)** *may result in a torque applied to the entire stack* **(b)** *which causes a corner of the stack to lift slightly off the ground* **(c)** *before settling down via an inelastic collision* **(d)**. *If we imagine trying to simulate this behavior* backward, *however, we notice that at step* **(d)** *before any collisions have occurred we must apply our inelastic collision to cause a nonzero velocity for the stack; failure to do so means that an opposing torque will be applied at step* **(b)** *that, unless it is somehow damped out through other low-probability events, will cause the stack to disintegrate almost immediately.*

are extremely prone to flying apart under the reverse simulation (this corresponds to a forward simulation where all the bodies arrive at their precise positions in the stack simultaneously, which can limit the range of motions). The cause of this phenomenon can be seen in Figure 5.8; basically, to ensure that the stack remains balanced when running backward, we must anticipate collisions before they occur and ensure that the stack has the exact velocity at the time of the collision to cancel out the impulse generated by the impact. Ensuring this for every impact would necessitate a substantial amount of look-ahead that would make backward simulation less practical. We take an alternative approach which is not physically valid but produces reasonable-looking motion: we allow the normal force to drop *below* 0 for objects that have out-degree greater than 1 in the collision graph. This corresponds to allowing stacked objects to attract each other, which prevents blocks at the bottom of the stack from transitioning from resting to bouncing.

5.3.4 Joints and constraints

One of the advantages of using the LCP formulation is that joints are easy to add; we simply add extra variables to $\mathbf{f}$ in (3.14) for the Lagrange multipliers (see Witkin's course notes for details [Wit01]). To generate backward simulations involving jointed and otherwise constrained bodies, this formulation holds without modification save reversing the signs on any Baumgarte stabilization terms (we want joint error to decrease as the simulation steps backward, not increase).

Despite this, generating *plausible* time-reversed simulations of articulated bodies remains challenging: for short periods, the generated motion is reasonable, but eventually the attaching bodies begin rebounding off each other with exponentially growing velocities. This is similar to what we expect if we simulate deformable bodies backward; because vibrations damp out in the forward direction, they must diverge to infinity in the backward direction. To understand why this problem arises in the first place, consider Figure 5.9; here, the only way to avoid a nonzero velocity at the joint (and hence inter-body collisions) is if we experience the low probability event that the simulation chooses to cause the resting bottom block to rise up off the surface at exactly the right moment.

One heuristic that can ameliorate the effect somewhat for articulated bodies is to set the coefficient of restitution to unity between bodies connected by a series of joints, but we recognize that this is a highly unappealing solution and hope that future work will suggest better alternatives.

For actuated joints such as those appearing characters, we have an additional degree of control that can be used to kill off the relative motion between two bodies at joints. As the character can apply whatever force across the joint that it "decides" to, we could simply choose to apply forces that tend to damp out relative motion during reverse simulation. This is similar to the backwards damping force described above. When the motion is played back in the forward direction, we will find that the character appears to accelerate into collision events, which may look more or less odd depending on the circumstances and the amount of damping that is applied.

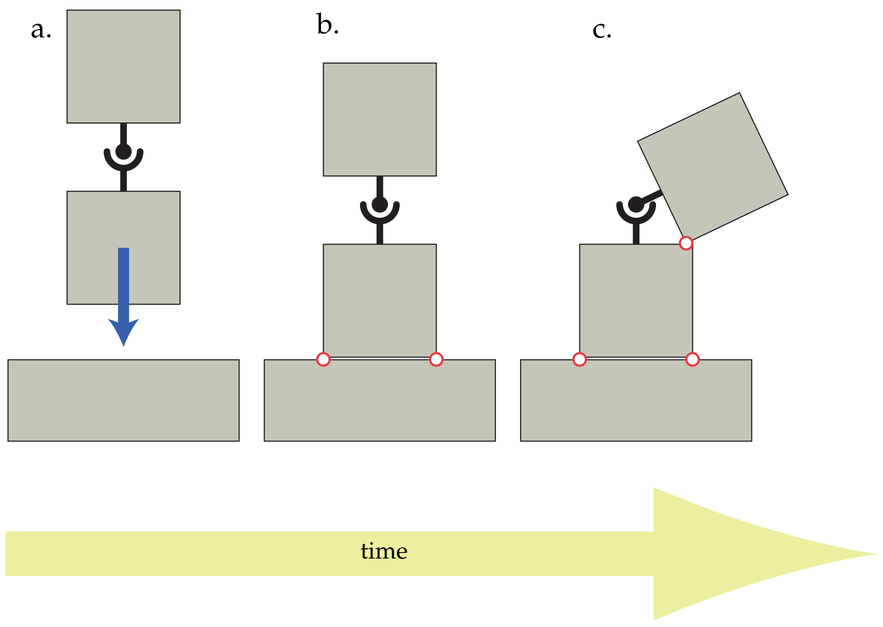


Figure 5.9: Issues with joints: *In a forward simulation, dropping these two connected bodies results in the resting configuration shown in* **(c)**. *If we begin our backward simulation from this resting configuration, however, we quickly run into problems; from rest, we must apply precisely the right impulse to the upper block to push it into the vertical configuration, and at that exact moment the bottom block must lift off the surface. If we fail to match this exactly, there will be some residual angular motion about the joint which will quickly get multiplied through inter-block collisions.*

5.3.5 Rolling

Rolling behavior is essential for generating realistic behavior for balls and cars. However, without a rolling friction model, it is impossible to generate rolling behavior during backward simulation *unless* the object is rolling without slipping in the initial conditions we provide to the solver. This is an unintuitive result, but it follows from the following two observations:

1. The principle of maximal dissipation ensures that any slipping grows without bound as the simulator steps in reverse. Suppose, for example, that our scene consists of a single sphere resting on the ground with a rotational velocity ω_i perpendicular to the page. In forward simulation, friction opposes this angular velocity, so in backward simulation it must reinforce it; the result is a sphere that spins ever faster while accelerating to the left.

2. In the absence of rolling friction, if an object is sitting at rest then there exists no forward simulation involving only rolling which could have concluded in that state. This means that we provide our backward simulator with the initial condition that our sphere is sitting unmoving then it is only through the application of rolling friction that we can start the ball rolling.

This means that a working rolling friction model is an essential part of backward simulation. Ours is quite simple: we add terms to the LCP problem that (1) oppose rotational motion and (2) are proportional to the normal force. By including them in the solver itself, we guarantee that (1) the rolling forces are proportional to the normal force, which would be difficult if we computed rolling friction before we solved for the normal forces, and (2) rolling friction does not cause violation of any of the other constraints, which would be difficult to guarantee if rolling friction was added in after the LCP solver step.

Our rolling friction model adds three rows to the LCP problem, one for each of the x-, y-, and z-directions. Denote the three directions by $\mathbf{q}_i$, $\mathbf{r}_i$, and $\mathbf{s}_i$ and let $\omega_{\mathbf{q}_i}^{t+\Delta t} = \mathbf{v}_i^{t+\Delta t} \cdot \mathbf{q}_i$ represent the projection of the angular velocity onto the vector $\mathbf{q}_i$. The model is nearly identical to friction; the force must be proportional to the normal force,

$$0 \le f_{\mathbf{q}_i} - \mu f_{N_i} \qquad \perp \qquad \omega_{\mathbf{q}_i}^{t+\Delta t} \tag{5.15}$$

$$0 \le f_{\mathbf{r}_i} - \mu f_{N_i} \qquad \perp \qquad \omega_{\mathbf{r}_i}^{t+\Delta t} \tag{5.16}$$

$$0 \le f_{\mathbf{s}_i} - \mu f_{N_i} \qquad \perp \qquad \omega_{\mathbf{s}_i}^{t+\Delta t} \tag{5.17}$$

subject to the constraints that it oppose angular velocity,

$$f_{\mathbf{q}_i} \cdot \omega_{\mathbf{q}_i}^{t+\Delta t} \le 0 \qquad f_{\mathbf{r}_i} \cdot \omega_{\mathbf{r}_i}^{t+\Delta t} \le 0 \qquad f_{\mathbf{s}_i} \cdot \omega_{\mathbf{s}_i}^{t+\Delta t} \le 0 \tag{5.18}$$

During reverse simulation, we allow the user to specify a goal rolling velocity ω^{goal}, much as they can for friction.

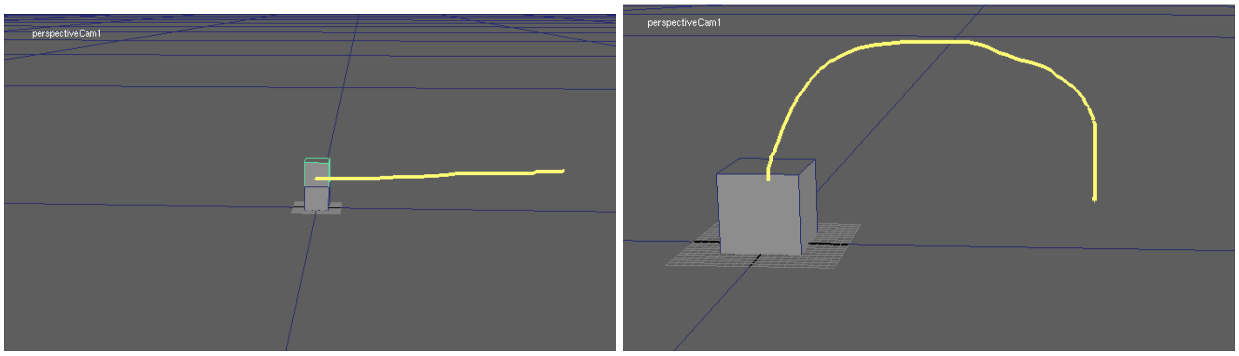


Figure 5.10: User-specified motion suggestions *can provide hints to the backward integrator to use when the resulting motion may be non-unique. An arced line indicates bouncing while a straight line indicates sliding with the direction of the line hinting the sliding direction. We use a simple method to distinguish between the two possibilities by fitting a 2nd-degree polynomial to the sketched curve and comparing the quadratic coefficient to a threshold.*

5.3.6 Sliding, skidding, and bouncing

When the initial conditions for the backward simulation specify that the object is at rest, we have an enormous amount of leeway in what the object should do. As described in section 5.2, we can provide hints to the simulator on a per-contact basis specifying that the object begin sliding or that a small nonzero normal impulse be applied.

One possibility is to give the user direct control over these possibilities. In our interactive system, the user can use gestures to distinguish between sliding and bouncing (see Figure 5.10).

If the user fails to provide any particular direction, the simulator itself must make these decisions, preferably in a suitably random fashion. In developing this, our goal is to generate motion that is as similar as possible to motion generated using a forward simulation. We therefore generated over 1000 forward simulations of a block landing on a plane, randomizing both the initial angular and linear velocities. We recorded the contact state at each step of the simulation. In figure 5.11, we have plotted the number of contacts recorded in each frame of the simulation (only a subset of the simulations are shown here).

Ideally, we would like our backward simulation to generate a plot that looks as much like this as possible for simulated motions. To this end, we use a simple state-space model that is based on Markov chains. Let η^t denote the number of active contact points at time t. For general meshes this should depend on the active set used in the LCP solver (those contacts $\mathbf{n}_i$ where $f_{N_i} > 0$) as most contact detection algorithms, particular those that operate on arbitrary meshes, tend to err on the side of caution and return a far larger set of contact points than the minimal set needed to prevent interpenetration. For simple problems involving collision primitives though it was sufficient to use the set of contacts returned by ODE's native collision detection code, as this is tuned (for performance reasons) to return the fewest contacts that still generate correct simulation.

During reverse timestepping, we are given the set of contacts at time t and want to determine how

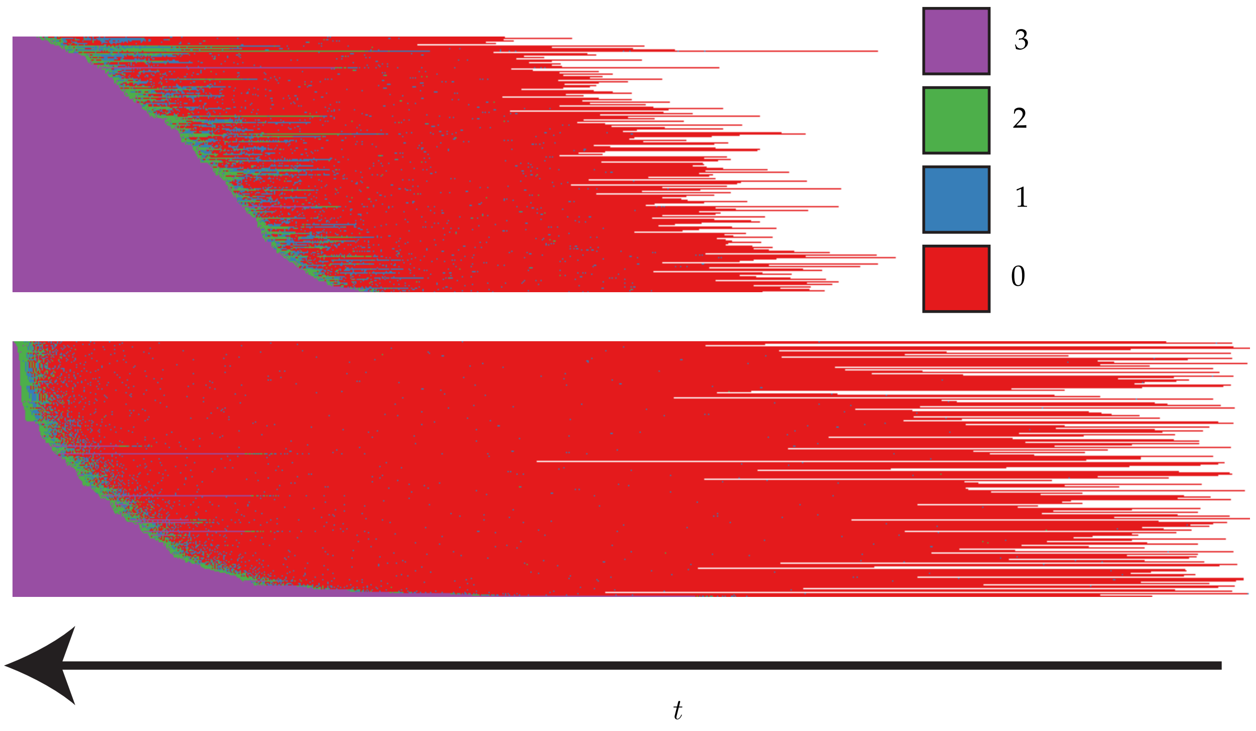


Figure 5.11: Contact state transitions: **Top:** *Contact state in rigid body simulation is complex, as we see transitions between* 0 *(free-flight),* 1*,* 2*, and* 3 *(flat on a face) contacts for a box thrown against a plane. Time runs from right to left, so at the far left of the plot the box is at rest; the simulations have been sorted lexicographically by contact configuration for easy viewing.* **Bottom:** *200 simulations computed using backward simulation; our goal is to get* qualitatively *similar simulations.*

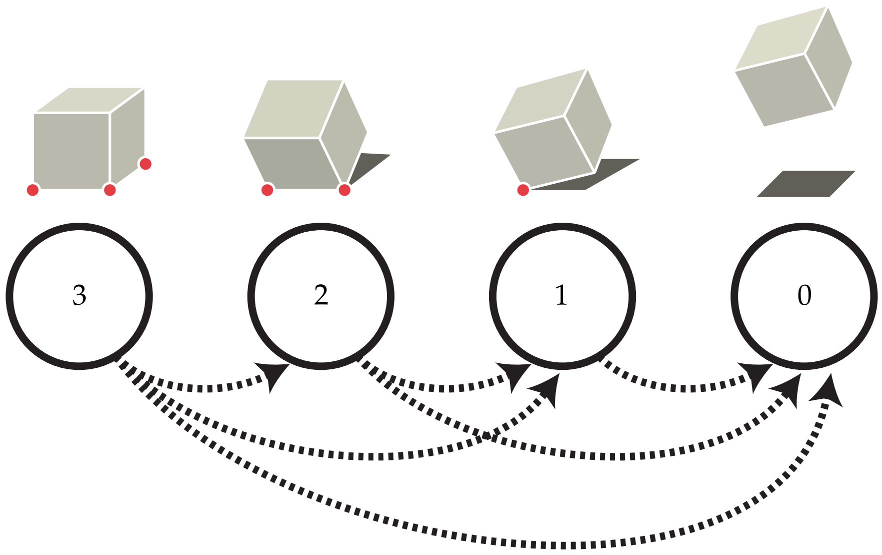


Figure 5.12: Contact transition model: *We use a simple model for determining when to transition between contact states during backward simulation. Because we can only apply impulses out of the plane, we can only transition to states with fewer contacts, as seen here. Probabilities for transitioning between contact states are computed from the data in Figure 5.11.*

many contacts should be active at time $t - \Delta t$. Our backward simulator can generate a nonzero velocity at any resting contact using the $v_{N_i}^{goal}$ parameter in (5.5), so if we choose $\eta^{t-\Delta t} \leq \eta^t$ we can always guarantee that at the end of the timestep, the resulting simulation will have at most $\eta^{t-\Delta t}$ contacts with zero relative velocity.

The Markov condition states that the distribution over possible contact states at time $t - \Delta t$ depends solely on the contact state at t,

$$\mathbb{P}(\eta^{t-\Delta t} = i \,|\, \eta^t, \eta^{t+\Delta t}, \ldots) = \mathbb{P}(\eta^{t-\Delta t} = i \,|\, \eta^t = j) \tag{5.19}$$

As there are only a small number of discrete possibilities for η^t, we can represent the probability of transitioning from $\eta^t = j$ to $\eta^{t-\Delta t} = i$ using the matrix,

$$\begin{pmatrix} \mathbb{P}(\eta^{t-\Delta t} = 1 \,|\, \eta^t = 1) & \mathbb{P}(\eta^{t-\Delta t} = 2 \,|\, \eta^t = 1) & \mathbb{P}(\eta^{t-\Delta t} = 3 \,|\, \eta^t = 1) & \cdots \\ \mathbb{P}(\eta^{t-\Delta t} = 1 \,|\, \eta^t = 2) & \mathbb{P}(\eta^{t-\Delta t} = 2 \,|\, \eta^t = 2) & \mathbb{P}(\eta^{t-\Delta t} = 3 \,|\, \eta^t = 2) & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{pmatrix} \tag{5.20}$$

We can measure these values directly from the data in Figure 5.11, as each timestep of forward simulation represents a transition from $\eta^{t-\Delta t}$ to η^t. To get probabilities for contact state transitions in reverse simulation, we need to reverse these quantities; each time we see a transition $\eta^{t-\Delta t} \rightarrow \eta^t$ during forward simulation, we record a transition $\eta^t \rightarrow \eta^{t-\Delta t}$ in our state transition matrix. To convert these to probabilities, we normalize by dividing each row by the sum of its entries. Because our system cannot model transitions from fewer contacts to more contacts (entries above

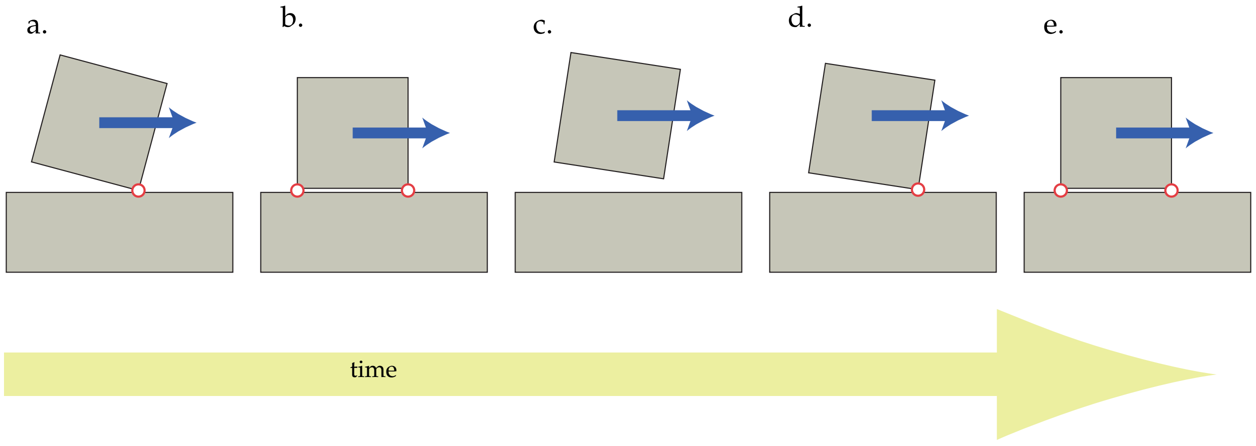


Figure 5.13: Contact state transitions that are impossible to generate during backward simulation. *Here, a block is sliding along one edge/vertex* **(a)** *when the opposing edge lands with sufficient velocity to generate an elastic response* **(b)**, *which causes the entire block to lift off the surface* **(c)** *before landing again with a totally inelastic response* **(d-e)**. *To generate this simulating backward, we would need to anticipate the landing at* **(b)** *at steps* **(c-d**) *in order to generate both the correct timings and magnitudes of impulses. Regardless, our reverse contact model is unable to generate this motion as* both *contacts at* **(b)** *would receive elastic collision responses.*

the diagonal), we simply set these to zero. Note that transition probabilities are highly dependent on the particular collision geometry used, but it would not be difficult to run a few dozen simple forward simulations with any provided geometry to infer the matrix (5.20).

During simulation, we follow the probabilistic finite state machine shown in Figure 5.12. At each timestep, our current contact state η^t tells us which row of (5.20) to index into. We generate a random number between 0 and 1 which tells us which contact state we will transition into. Once we know $\eta^{t-\Delta t}$, we need merely to select *which* contacts should be separated and set $v_{N_i}^{goal}$ appropriately. The result is far from perfect; the most obvious problem is the lack of transitions (during reverse simulation) from fewer contacts to more contacts. The corresponding transition from more to fewer contacts is common during forward simulation; see Figure 5.13 for one such example. Our reverse contact model is unable to produce a similar effect due to the lack of inelastic collisions. However, while there are significant differences between the plots, if we examine the resulting motion it seems plausible; in particular, the $t \to -t$ example below uses this model exclusively.

5.4 Results

To test the usefulness and scalability of our approach, we applied it to several different examples. For each of these (except the spelling balls example, which was only run once), we found a significant amount of variability from simulation to simulation, with some looking fairly reasonable and others looking highly unrealistic due to rapid energy gain or large angular velocities. Additionally, some simulations were more "interesting" than others, including, for example, tighter interactions between objects. We tended to find that looking at 10-20 example simulations in the Many-Worlds Browser produced at least one that was usable.

5.4.1 Spelling balls

Chenney and Forsyth [CF00] showed several examples of dropping balls into grid cells to spell out words. This is a nearly ideal example for our approach, because we care much more about the resulting shape than we do about exactly where the balls come from. The trade-off we make here is that we have very little control over where the balls fall from while Chenney and Forsyth incorporated this into their prior. In our version we dropped 3037 balls into square bins to spell out the classic typesetting dummy text "Lorem ipsum dolor sit amet…" (see Figure 5.14). Although our example has 100 times as many balls as the ACM example, it runs in just under an hour on our test machine, which has a 2.66GHz Intel Core 2 processor and 2GB of RAM. Performance could be improved by careful tuning of the broad-phase collision handling, but we didn't put any effort into this. Note that in this example good modeling of rolling and sliding is essential to getting realistic behavior toward the end of the simulation. One problem that we observe in this simulation is that more balls than we might expect are sliding into rather than rolling into place; this suggests that we need to tune the relative probabilities of sliding versus rolling, which is currently much more error-prone than it needs to be.

5.4.2 Spelling boxes

As we are running the simulation backward, we can make the final state as unrealistic as we want. In this case, we spell "$t \rightarrow -t$" without using bins to catch the objects (see Figure 5.15). Note that there is no requirement that the objects' motion be independent; in fact, many of the objects interact extensively. This is the best example for seeing the results of our contact state transition model (section §5.3.6), because all motion was computed at random rather than with user guidance. This example was computed in under a minute and exhibits some of the angular velocity growth that we noted in section 5.3; here we are applying a reverse damping force to tame velocity growth somewhat. The other problem that we see fairly frequently in this example is the rattling behavior described in Figure 5.6. As the state transition model has no knowledge of what other objects in the scene are doing, it may decide that a block in the middle of a group of blocks should start sliding, leading to rapid energy gain for the entire group. Provided the group isn't too large, this isn't too much of a problem, and can even produce appealing (albeit unlikely) behaviors such as two blocks sliding in from opposite directions before striking each other and stopping. As the density of blocks in the plane grows, however, energy gain can quickly become a distraction.

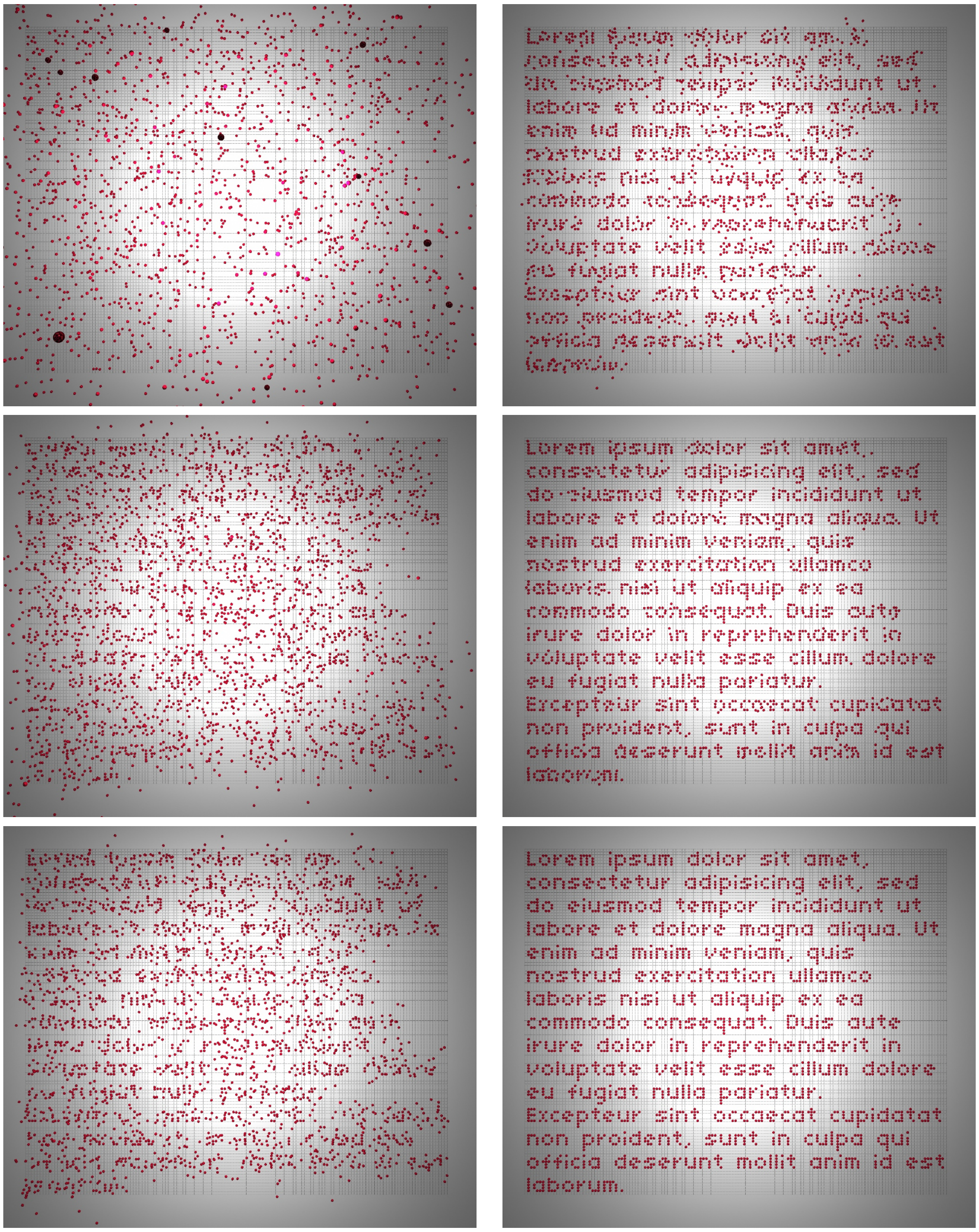


Figure 5.14: *3037 balls spell out the classic nonsense phrase "lorum ipsum dolor sit amet..."*

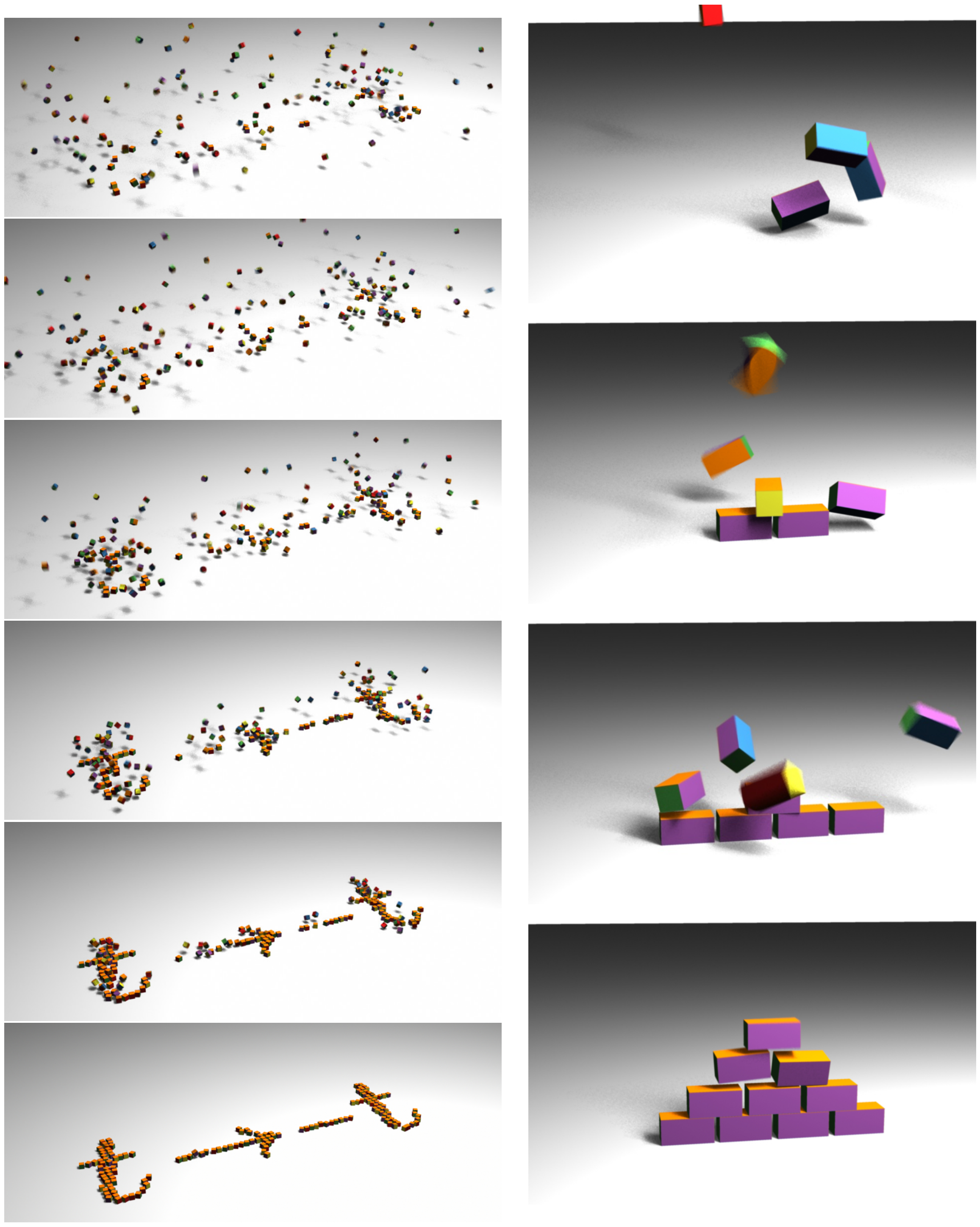


Figure 5.15: Left: *Boxes spell out the time-reversal formula* $t \rightarrow -t$. **Right:** *10 boxes bounce in and form a stack.*

5.4.3 Stacking

We built this stack out of 10 bricks and simulated it using the stacking heuristics described in §5.3.3. Despite these heuristics, this simulation was extremely prone to rapid velocity growth due to rattling, which made many of the resulting simulations look rather unrealistic, and these had to be rejected.

5.4.4 Collision scenario design

Backward simulation can also be useful for specifying intermediate frames of a simulation. For example, the collision scenario shown in Figure 5.16 was simulated by using the intermediate frame as initial conditions for both forward and backward rigid body simulators. These objects appear to fly in, collide and bounce up, spelling "CRASH," then fall down onto the ground. The primary visual artifact we see in this example is that the boxes seem particularly prone to angular velocity growth, which is due in part to their nonuniform scaling. The result is that boxes, rather than falling from the sky, skittle in from the side with large angular velocities. To counter this somewhat we have applied the reverse angular damping force (5.14); unfortunately, this means that energy is lost running backward, which further amplifies the perception that the objects do not seem to be losing energy in collisions the way we might expect.

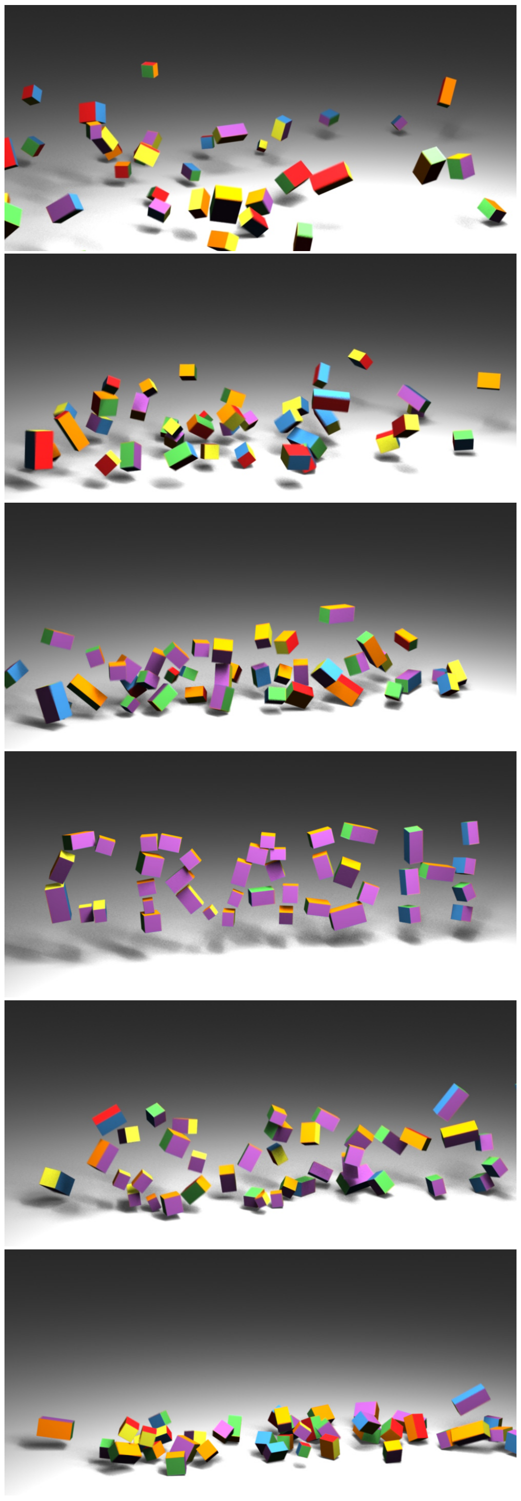


Figure 5.16: Collision scenario *designed by one intermediate frame.*

Chapter 6

Conclusion

Rigid bodies are ideal for many simulation problems encountered in real-world production due to their speed, stability, and robustness. Shattering [OH99, BHTF07], explosions, car crashes, rag doll characters, tentacles [CDH06] and a variety of other phenomena can all be simulated using rigid bodies with contact and joint constraints. And while much of the early work in controlling passive dynamics focused on the simulation of rigid bodies, there has been little follow-up work since, even as great strides have been made in controlling fluids and deformable objects. Despite the higher cost of simulation, the many more degrees of freedom and (in most cases) differentiability of the simulation engines make the solution to constraint problems in these other domains a feasible albeit difficult task. Unfortunately, faster computers and more sophisticated methods for computing gradients cannot help us to solve problems in a domain where each colliding object carves out distinct discontinuities in state space.

This thesis attempts to fill that gap using methods that are quite simple to both understand and implement. While Many-Worlds Browsing is not a panacea, it has succeeded in generalizing control methods to some domains (articulated characters, longer simulations) where previous methods struggled. It is worth taking a step back and considering whether we can take away any broader lessons that can be generalized to other domains.

It is important as researchers to remember that a method's utility and practicality does not necessarily correlate with its sophistication. The failure of many optimization techniques to infiltrate day-to-day production is in many cases *because* of the mathematical sophistication that is required not just during implementation but (despite our best efforts) during the application to specific problems. Any technique that relies on regular care and feeding by highly-specialized and difficult to find experts will need to provide truly spectacular results to be even considered. It is a truism that maintenance occupies more of the life-cycle than the initial roll-out does; methods that can be decoupled from the underlying simulation will prove less fragile in the long run. Hence, we see widespread success of control methods based on little more than force fields and damped springs.

If we want to develop methods that will be adopted, we must present a compelling improvement over the alternative. This means considering not just algorithms but the entire pipeline end-to-end. It is acceptable for an optimization method to run overnight, but only if the user's interface for specifying the optimization problem is all but foolproof; we are not making things easier if we turn the search over simulation parameters into an even more complicated one over optimization parameters. In many cases, we must compete not against other control methods but against key-framing: if an object's motion is important enough to run through an overnight optimization run, it is likely that it would be worth the hours of animator time that hand-animation would require. User interfaces are important when the decision to use a tool will be up to non-experts; I would argue that much of the excitement that Popović and colleagues generated [PSE$^+$00] was due more to the interface than to the sophisticated mathematics that powered it. For fluid control, the enormous strides made in back-end algorithms have long outstripped the key-framing interfaces that are still used to drive them, meaning they can really only be used to drive characters made of fluid [REN$^+$04, ABC$^+$07], which belies much of their initial promise.

It is thus important to always keep in mind who our theoretical end users are. By keeping the users perpetually in the loop, providing useful feedback, ensuring that interfaces are both intuitive and discoverable, we maximize the probability that our methods will see wider adoption. It is less likely that the "best" result will arrive as the result of a single grand minimization than a series of smaller steps through which the user can explore the possibilities. Any assessment of the trade-offs between control and realism is necessarily one that only the user can make, and is likely to vary from simulation to simulation and day to day. Thus, a tool which allows the user to iterate toward a solution will be more usable than one which forces the user to specify everything *a priori*.

We hope that this work will inspire future methods. There are many scenarios that neither Many-Worlds Browsing nor backward simulation can efficiently solve. Much work remains if we have any hope of applying MWB in other problem domains, such as cloth or fluids, where minimizing the amount of stored data will be of the highest priority. Certainly even the browsing of rigid bodies could be improved in any number of ways, including better metrics and query methods and more sophisticated search primitives.

6.1 Reverse simulation

Reverse simulation, originally intended as a simple extension to the Many-Worlds Browsing framework, has proven surprisingly complicated. The issues that arise in the process are both subtle and complex, and while we made an effort to highlight as many specific limitations as we could, there are other more general problems that are harder to characterize concretely.

As an example, take the Pachinko machine example from the Many-Worlds Browsing chapter. This seems like an obvious place to use backward simulation, as we care much more about the ending state of the simulation than what paths the letters take. As there are no joints or stacking in the simulation, the limitations we discuss in Chapter 5 do not seem to apply. If we actually set up the simulation and run it backward, however, we are sorely disappointed by the results. Letters spend the entire simulation rattling back and forth between the bottom-most pegs before

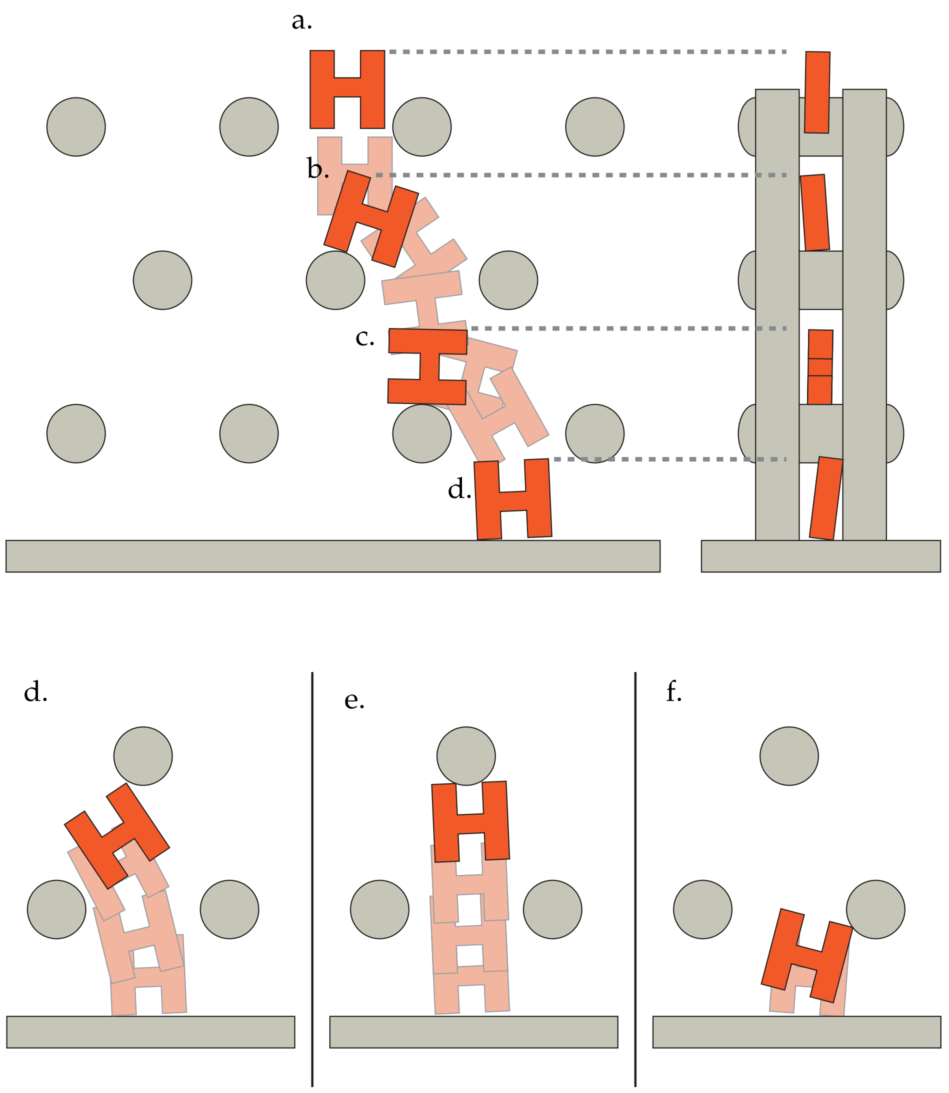


Figure 6.1: Backwards Pachinko machine? *When we run the Pachinko machine forward in time, we find many events where the object either bounces lightly off the top of a peg* **(b)** *or balances briefly on the peg before rolling off* **(c)***. We may also find that it rests for periods of time against one sheet of glass or the other* **(top right)***, depending on interactions with the pegs. If we try to run the machine backward, we first discover that rattling between the glass panes is a problem; any time the object strikes a pane going backwards, we must find another impulse in the system to justify why (in the forward direction) it would have come off the surface. If we address this problem, by (for example) removing the panes and using an in-plane constraint instead, we find that events which are unusual during forward simulation such as striking the bottom of a peg* **(e)**-**(f)** *become quite common during backward simulation leading to high energy gain. Likewise, events which would be common during forward simulation (rolling gently off a peg) are rare during reverse simulation as the velocities must be such that the object comes barely to rest upon the top.*

gaining enough energy to simply tunnel through the walls. Simulations where the letters are able to reach above the bottom-most rows of pegs prove vanishingly rare in practice. As users, we are likely to be surprised and frustrated by our total inability to reach our goals, and it is really this issue (more than the specific restrictions on joints and stacking) that must be tackled before this technique will become routinely usable.

Looking more closely at the forward simulation as in Figure 6.1, we can pinpoint at least one reason our reverse simulator is unable to generate acceptable motion. During forward simulation, the events that we see frequently, like rolling off the top of a peg, are considerably less probably from the reverse simulator's point of view. To reach the top of a peg with a small velocity, for example, the reverse-simulated object must have an arc threading neatly in between the other pegs in the system; to roll off a peg, our object would need to reach the peg with a relative velocity of essentially 0. Finding a way to the top of the Pachinko machine requires chaining a whole series of improbable events together. On the other hand, we find that virtually any forward simulation is capable of guiding the object from the top to the bottom of the machine.

We can imagine a variety of strategies that might expand the set of problems that could be solved with backward simulation. The first is to simply switch to forward simulation in cases where it produces better results. The decision would need to be automated in some fashion as expecting users to have the kind of intuition necessary to decide between the two methods is almost certainly not realistic. Reasoning about which method would be better is a global problem, though, and possibly harder than the original control problem. A simpler approach might involve asking users to specify both start and end states and running simulations from both directions, as our Many-Worlds Browsing work indicates that presenting the user with plenty of data tends to pay off.

Using such a strategy, however, we will tend to find that even for scenes where backward simulation works well, many of the individual motion examples generated will appear implausible. Our experience with Many-Worlds Browsing suggests that polluting the data set with poor-quality motion will discourage users. Our own ability to quickly pinpoint simulations that look unrealistic gives us some hope that maybe we could develop an algorithmic test for determining whether a simulation is acceptable. Such a test could be used to notify us when a simulation veers into implausibility, potentially saving valuable computing time, or if invertible could "guide" simulations into more plausible regions of the parameter space (conversely, if no such algorithm existed, reverse simulation would make a rather unconventional CAPTCHA [vABHL03]). We have noted before that rapid energy growth and high angular velocities are common characteristics of backward simulations that can lead to an implausible appearance, but while both are simple to detect it seems likely that other, more global metrics may be necessary. Knowing that we expect objects to strike pegs from above, for example, might be important, but it is not clear how to generalize these kinds of metrics. One possibility is to make the determination on a per-scene level; we could run a few thousand copies of the same simulation and then learn a classifier to determine what are plausible or implausible animations; see for example work by Ren and colleagues applying a similar method to motion capture data [RPE+05]. Potential problems with this approach include the fact that we only have positive examples to work with and it is still not clear what the features should be and how they should be computed.

6.1.1 Integrating forward and backward simulation

Another approach that seems promising is to integrate backward and forward simulation more closely together. There are many potential variations on this theme. The simplest is a variation on the Many-Worlds Browsing framework: we could use forward simulation for certain objects in the scene and backward simulation for others. The capabilities we described in our refinement section would ensure that motion remains physical: if we added a backward-simulated object to a scene which was originally run using forward simulation, our refinement framework would activate objects as needed in order to ensure physical correctness. There is some risk though that combining the unintuitive nature of backward simulation with the need to juggle which objects should be forward-simulated or which should be reverse-simulated will be too complicated for all but the most dedicated of users to learn.

One potential avenue for future work is to combine forward and backward simulation to solve problems where both the beginning and end state are specified (the *boundary-value problem*). In its most basic form we would simulate the object forward from the start state and backward from the end state and attempt to connect the two simulations somewhere in the middle. As it is extremely unlikely that the simulations will actually match up at some point in the middle, some form of blending will be necessary. For plausibility, the blending should respect contact constraints [JTCW]; alternatively, an approach similar to Popović and colleagues [PSE$^+$00] could be used to drive the two simulations closer together while minimizing the magnitude of the applied perturbations, as gradient-based techniques work extremely well on small self-contained problems (e.g., the three to four collisions surrounding the blend time). The main difficulty of this approach lies in finding the two simulation states (generated by different simulators) that are sufficiently close to effect a plausible blend. This is an expensive search, comparing each forward simulation against each backward simulation, and will necessitate the use of acceleration structures like the ball tree [Uhl91] or cover tree [BKL06]. It is also impossible for a given pair of start/end states to guarantee that a transition will exist from one to the other in the space of plausible forward/backward simulations, and the approach will have difficulty scaling up to larger numbers of objects unless each object can be treated independently.

Nonetheless, this is a potential starting point for finding ways to solve challenging boundary-value problems. We can start to imagine variants on the basic method such as *multiple shooting* [PSE03, TMPS03] in which we can subdivide the time range into smaller parts, trying to get each segment to match up at the boundary. This has a number of advantages; most relevant to rigid bodies is the fact that in a shorter time period the simulation has fewer opportunities to stray, making finding plausible connections potentially easier. Such a method would have many similarities to the multiple-shooting technique described by Popović and colleagues, except that reverse simulation combined with random sampling would guard against local minima, playing the same role that a good initial guess provided in the earlier work.

Alternatively, instead of trying to solve the problem globally, we could imagine using boundary-value techniques to try to address some of the limitations of backward simulation. Issues with stacking in particular could be mostly eliminated if we simulated blocks of frames rather than a single frame at a time. Consider again Figure 5.8; the time between (b) and (d) is rather minimal, and only a relatively short time horizon would be necessary to be able to anticipate when simulating frame (d) that we would need a "lift-off" in order to correctly account for the collision at frame (b). Thus, we could consider replacing our single-stepping backward simulator with one that uses a boundary-value framework like the one described in the previous paragraph to simulate short clips of motion and ensure that they connect up. Intuitively, as long as we get the bricks "close" to the right ending (stacked) configuration, they will naturally settle into place using forward simulation. The missing ingredient though is how to choose an earlier (starting) state. While we know how the bricks should end up (i.e., stacked), the question of where they came from is normally answered by our single-stepping reverse timestepping scheme; plugging this into a boundary-value solver will just produce the same bad results as before. Hence, heuristics may be necessary to project likely earlier-time locations for the bricks.

We might similarly ask whether it is possible to use a longer time horizon to address other plausibility issues in backward simulation. For example, large angular velocities result from large angular moments imparted when the angle between the radius vector and the contact normal is near ninety degrees. However, this angle is a function of the contact geometry and if we only notice at the current frame the we are going to be increasing the angular velocity it is too late to change the direction of the impulse. By looking at frames around the current frame, we could potentially notice that a change in the object's path would produce more favorable conditions. Similarly, the ability to produce a nonzero velocity at a collision without applying forces would require that we perform a search for other contacts or forces in the system that could explain the nonzero velocity; this might mean making small adjustments to the objects velocity at other frames to change the collision geometry at the current frame.

6.1.2 Plausibility

The existence of a process (backward simulation) which can produce nearly endless runs of physically correct but implausible motion calls into question our "local" notion of plausibility. Recall that we defined plausibility in terms of perturbations at collisions: if the object's velocity is perturbed by less than the threshold value at each collision, we consider the entire simulation plausible. The justification is that existing studies of simulation plausibility take only single collisions into account, and we do not know how collisions might interact. Barzel and colleagues [BHW96] do cite the example of a simulation that is perturbed just the right amount to produce an unexpected result. We noted that the likelihood of such a series of perturbations under our sampling scheme was vanishingly slow. However, reverse simulation can produce very implausible-looking simulations even in the absence of perturbations. This includes not only the issues discussed earlier but also more general unlikely events; e.g., an object might spin in rapidly from off camera but strike the ground at just the right point to completely cancel its angular velocity, which ends up looking somewhat magical.

Quantifying this kind of improbability is non-obvious, however. For one thing, our original aim was to generate events that are improbable. Given a scene where a group of bricks come together and just "happen" to form a tower, how are we to determine whether this happened in a "plausible" fashion? For another, this kind of plausibility is difficult to get a handle on. Intuitively, the idea of "ordered-ness" seems to line up with the notion of entropy in information theory, but it can be difficult to make that theory apply. Suppose for example that we drop a bunch of blocks flat on the ground and they happen to spell out the text of, say, *Hamlet* (in a similar fashion to our $t \rightarrow -t$ example). Obviously, we would consider this an extremely fortunate event. Now, suppose that we drop them and they end up in another configuration where they do not spell out anything. If we were to run many such simulations, we might well discover that if we looked at the distribution of states the second simulation is every bit as probable as the first; after all, if we pick any *particular* unordered state, it will be highly improbable, it is only by summing over all such unordered states that we find a probability approaching 1. Put another way, each individual box is in a relatively probable state, it is the combination of all the boxes being in the "right" places that we find unlikely. In our *Hamlet* case, furthermore, is it really the fault of the backward simulator that the resulting motion looks implausible given that the implausible final state was given as input to the algorithm?

While the presented backward simulation is burdened with enough limitations that it seems not quite ready for the production pipeline, we hope that by being upfront about these issues we will spur future work in the area. Certainly the area of reverse simulation seems completely wide open and we hope that in the future it will find its way into the artist's toolbox.

Bibliography

[ABC$^+$07] Christoph Ammann, Doug Bloom, Jonathan M. Cohen, John Courte, Lucio Flores, Sho Hasegawa, Nikos Kalaitzidis, Terrance Tornberg, Laurence Treweek, Bob Winter, and Chris Yang. The birth of Sandman. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 26, New York, NY, USA, 2007. ACM.

[AM00a] Marc Alexa and Wolfgang Müller. Representing animations by principal components. *Computer Graphics Forum*, 19(3):411–418, August 2000.

[AM00b] Ulf Assarsson and Tomas Möller. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools*, 5(1):9–22, 2000.

[ANSN06] Alexis Angelidis, Fabrice Neyret, Karan Singh, and Derek Nowrouzezahrai. A controllable, fast and stable basis for vortex based smoke simulation. In *Proc. 2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 25–32, September 2006.

[AP96] M. Anitescu and F. A. Potra. Formulating dynamic multi-rigid-body contact problems with friction as solvable Linear Complementarity Problems. *Nonlinear Dynamics*, 14(3):231–247, October 1996.

[AP98] Uri M. Ascher and Linda R. Petzold. *Computer Methods for Ordinary Differential Equations and Differential-Algebraic Equations*. SIAM: Society for Industrial and Applied Mathematics, July 1998.

[Ari06] Okan Arikan. Compression of motion capture databases. *ACM Trans. on Graphics*, 25(3):890–897, 2006.

[Bar91] David Baraff. Coping with friction for non-penetrating rigid body simulation. In *Computer Graphics (Proc. SIGGRAPH 91)*, pages 31–40, July 1991.

[Bar94] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proc. of ACM SIGGRAPH 1994*, pages 23–34, July 1994.

[Bar96] David Baraff. Linear-time dynamics using Lagrange multipliers. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 137–146, August 1996.

[Bar01] David Baraff. Rigid body simulation. In *Physically Based Modeling: SIGGRAPH 2001 Course 25*, 2001.

[Ben99] Jon Bentley. *Programming Pearls*. Addison-Wesley Professional, second edition edition, 1999.

[BHTF07] Zhaosheng Bao, Jeong-Mo Hong, Joseph Teran, and Ronald P. Fedkiw. Fracturing rigid materials. *IEEE Transactions on Visualization and Computer Graphics*, 13(2), March 2007.

[BHW94] David E. Breen, Donald H. House, and Michael J. Wozny. Predicting the drape of woven cloth using interacting particles. In *Proceedings of SIGGRAPH 94*, Computer Graphics Proceedings, Annual Conference Series, pages 365–372, July 1994.

[BHW96] Ronen Barzel, John F. Hughes, and Daniel Wood. Plausible motion simulation for computer animation. In *EGCAS '96: Seventh International Workshop on Computer Animation and Simulation*, 1996.

[BKL06] Alina Beygelzimer, Sham Kakade, and John Langford. Cover tree for nearest neighbor calculations. In *Proc. of the 23rd International Conference on Machine Learning*, June 2006.

[BMF03] R. Bridson, S. Marino, and R. Fedkiw. Simulation of clothing with folds and wrinkles. In *Proc. 2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 28–36, August 2003.

[BMWG07] Mikls Bergou, Saurabh Mathur, Max Wardetzky, and Eitan Grinspun. TRACKS: Toward directable thin shells. *ACM Transactions on Graphics*, 26(3):50:1–50:10, July 2007.

[BO04] Gareth Bradshaw and Carol O'Sullivan. Adaptive medial-axis approximation for sphere-tree construction. *ACM Transactions on Graphics*, 23(1):1–29, 2004.

[BPP01] Philippe Beaudoin, Sèbastien Paquet, and Pierre Poulin. Realistic and controllable fire simulation. In *Proc. of Graphics Interface 2001*, pages 159–166, Toronto, Ontario, 2001. Canadian Information Processing Society.

[BS94] Karl-Heinz Brandenburg and Gerhard Stoll. ISO-MPEG-1 audio: A generic standard for coding of high-quality digital audio. *Journal of the Audio Engineering Society*, 42(10):780–792, October 1994.

[BSM$^+$03] Hector M. Briceño, Pedro V. Sander, Leonard McMillan, Steven Gortler, and Hugues Hoppe. Geometry videos: a new representation for 3D animations. In *Proc. 2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 136–146, August 2003.

[BTH$^+$03] Kiran S. Bhat, Christopher D. Twigg, Jessica K. Hodgins, Pradeep K. Khosla, Zoran Popović, and Steven M. Seitz. Estimating cloth simulation parameters from video. In *Proc. 2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 37–51, July 2003.

[BW98] David Baraff and Andrew P. Witkin. Large steps in cloth simulation. In *Proc. of ACM SIGGRAPH 1998*, pages 43–54, July 1998.

[CDH06] Brice Criswell, Karin Derlich, and Don Hatch. Davy Jones' beard: rigid tentacle simulation. In *SIGGRAPH '06: ACM SIGGRAPH 2006 Sketches*, page 117, New York, NY, USA, 2006. ACM.

[CF99] Kin-Pong Chan and Ada Wai-Chee Fu. Efficient time series matching by wavelets. In *Proc. of the Fifteenth International Conference on Data Engineering*, pages 126–133, 1999.

[CF00] Stephen Chenney and David A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proc. of ACM SIGGRAPH 2000*, pages 219–228, July 2000.

[CGC$^+$02] Steve Capell, Seth Green, Brian Curless, Tom Duchamp, and Zoran Popović. Interactive skeleton-driven dynamic deformations. *ACM Trans. on Graphics*, 21(3):586–593, July 2002.

[CGWS92] J. A. Crowe, N. M. Gibson, M. S. Woolfson, and M. G. Somekh. Wavelet transform as a potential tool for ECG analysis and compression. *J. Biomed. Eng.*, 14:268–272, May 1992.

[Coh92] Michael F. Cohen. Interactive spacetime control for animation. In *Computer Graphics (Proc. of SIGGRAPH 92)*, pages 293–302, July 1992.

[CPS92] Richard W. Cottle, Jong-Shi Pang, and Richard E. Stone. *The Linear Complementarity Problem*. Academic Press, 1992.

[CZ92] David T. Chen and David Zeltzer. Pump it up: computer animation of a biomechanically based model of muscle using the finite element method. In *Computer Graphics (Proc. of SIGGRAPH 92)*, pages 89–98, 1992.

[Dav06] Timothy A. Davis. *Direct Methods for Sparse Linear Systems*. SIAM, 2006.

[DCHH88] Jack J. Dongarra, Jeremy Du Croz, Sven Hammarling, and Richard J. Hanson. An extended set of FORTRAN basic linear algebra subprograms. *ACM Trans. Math. Softw.*, 14(1):1–17, 1988.

[FF01] Nick Foster and Ronald Fedkiw. Practical animation of liquids. In *Proc. of ACM SIGGRAPH 2001*, pages 23–30, August 2001.

[Fit54] Paul M. Fitts. The information capacity of the human motor system in controlling the amplitude of movement. *Journal of Experimental Psychology*, 47(6):381–391, June 1954.

[FL04] Raanan Fattal and Dani Lischinski. Target-driven smoke animation. *ACM Trans. on Graphics*, 23(3):441–448, August 2004.

[FM97] Nick Foster and Dimitri Metaxas. Controlling fluid animation. In *Proc. CGI '97*, pages 178–188, 1997.

[Fra03] Keir Fraser. *Practical Lock-Freedom*. PhD thesis, University of Cambridge, September 2003.

[GBD$^+$94] Al Geist, Adam Beguelin, Jack Dongarra, Weicheng Jiang, Robert Manchek, and Vaidyalingam S. Sunderam. *PVM: Parallel Virtual Machine: A Users' Guide and Tutorial for Network Parallel Computing*. MIT Press, November 1994.

[GBF03] Eran Guendelman, Robert Bridson, and Ronald P. Fedkiw. Nonconvex rigid bodies with stacking. *ACM Trans. on Graphics*, 22(3):871–878, 2003.

[GHF$^+$07] Rony Goldenthal, David Harmon, Raanan Fattal, Michel Bercovier, and Eitan Grinspun. Efficient simulation of inextensible cloth. *ACM Transactions on Graphics*, 26(3):49:1–49:7, July 2007.

[GK04a] C. W. Gear and Ioannis G. Kevrekidis. Computing in the past with forward integration. *Physics Letters A*, 321(5):332–343, February 2004.

[GK04b] Igor Guskov and Andrei Khodakovsky. Wavelet compression of parametrically coherent mesh sequences. In *Proc. 2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 183–192, July 2004.

[GL96] Gene H. Golub and Charles F. Van Loan. *Matrix Computations*. The Johns Hopkins University Press, third edition edition, 1996.

[GLM96] Stefan Gottschalk, Ming Lin, and Dinesh Manocha. OBB-Tree: A hierarchical structure for rapid interference detection. In *Proceedings of SIGGRAPH 96*, Computer Graphics Proceedings, Annual Conference Series, pages 171–180, August 1996.

[Gra98] F. Sebastian Grassia. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools*, 3(3):29–48, 1998.

[GTH98] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. NeuroAnimator: Fast neural network emulation and control of physics-based models. In *Proc. of ACM SIGGRAPH 1998*, pages 9–20, July 1998.

[Hah88] James K. Hahn. Realistic animation of rigid bodies. In *Computer Graphics (Proc. of SIGGRAPH 88)*, pages 299–308, August 1988.

[HC75] K.H. Hunt and F.R.E. Crossley. Coefficient of restitution interpreted as damping in vibroimpact. *Trans. ASME, Journal of Applied Mechanics*, pages 440–445, 1975.

[HHH97] D. Haugland, J. G. Heber, and J. H. Husøy. Optimisation algorithms for ECG data compression. *Medical and Biological Engineering and Computing*, 35:420–424, 1997.

[Hil97] M. L. Hilton. Wavelet and wavelet packet compression of electrocardiograms. *IEEE Trans. of Biomedical Engineering*, 44(5):394–402, 1997.

[Hru08] Joel Hruska. NVIDIA snaps up physics processing company Ageia. `http://arstechnica.com/news.ars/post/`

`20080204-nvidia-snaps-up-physics-processing-company-ageia.html`, February 2008.

[HS04] Harry Hochheiser and Ben Shneiderman. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Information Visualization*, 3:1–18, 2004.

[IR03] Lawrence Ibarria and Jarek Rossignac. Dynapack: space-time compression of the 3D animations of triangle meshes with fixed connectivity. In *Proc. 2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 126–135, August 2003.

[JT05] Doug L. James and Christopher D. Twigg. Skinning mesh animations. *ACM Trans. on Graphics*, 24(3):399–407, August 2005.

[JTCW] Doug L. James, Christopher D. Twigg, Andrew A. Cove, and Robert Y. Wang. Mesh Ensemble Motion Graphs: Data-driven mesh animation with constraints. In submission.

[KANB03] Zoran Kačić-Alesić, Marcus Nordenstam, and David Bullock. A practical dynamics system. In *SCA '03: Proceedings of the 2003 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pages 7–16, Aire-la-Ville, Switzerland, Switzerland, 2003. Eurographics Association.

[KE05] P. Krysl and L. Endres. Explicit Newmark/Verlet algorithm for time integration of the rotational dynamics of rigid bodies. *International Journal for Numerical Methods in Engineering*, 62(15):2154–2177, April 2005.

[Kem03] William E. Kempf. Boost.Thread. In Beman Dawes, editor, *Boost Libraries Documentation*. 2003.

[KEP05] Danny M. Kaufman, Timothy Edmunds, and Dinesh K. Pai. Fast frictional dynamics for rigid bodies. *ACM Trans. on Graphics*, 24(3):946–956, August 2005.

[KG06] Scott Kircher and Michael Garland. Editing arbitrarily deforming surface animations. *ACM Trans. on Graphics*, 25(3):1098–1107, 2006.

[KKiA05] Ryo Kondo, Takashi Kanai, and Ken ichi Anjyo. Directable animation of elastic objects. In *Proc. 2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 127–134, July 2005.

[KT07] Markus Kurtz and Jerry Tessendorf. Simulation, simulation, simulation: integration of multiple simulation techniques to create realistic water motion in order to maintain highest level of flexibility. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 94, New York, NY, USA, 2007. ACM.

[KYT$^+$06] Liliya Kharevych, Weiwei Yang, Yiying Tong, Eva Kanso, Jerrold E. Marsden, Peter Schröder, and Matthieu Desbrun. Geometric, variational integrators for computer animation. In *2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 43–52, September 2006.

[LA06] Jyh-Ming Lien and Nancy M. Amato. Approximate convex decomposition of polyhedra. Technical Report TR06-002, Texas A&M University, December 2006.

[Len99] Jerome Edward Lengyel. Compression of time-dependent geometry. In *SI3D '99: Proc. of the 1999 symposium on Interactive 3D graphics*, pages 89–95, New York, NY, USA, 1999. ACM Press.

[LF02] Arnauld Lamorlette and Nick Foster. Structural modeling of natural flames. *ACM Trans. on Graphics*, 21(3):729–735, July 2002.

[LH87] Charles L. Lawson and Richard J. Hanson. *Solving Least Squares Problems*. SIAM, 1987.

[Lit87] Michael Litzkow. Remote Unix - turning idle workstations into cycle servers. In *Usenix Summer Conference*, pages 381–384, 1987.

[LKK01] Mark Last, Yaron Klein, and Abraham Kandel. Knowledge discovery in time series databases. *IEEE Trans. on Systems, Man and Cybernetics, Part B*, 31(1):160–169, 2001.

[LN94] H. M. Lankarani and P. E. Nikravesh. Continuous contact force models for impact analysis in multibody systems. *Nonlinear Dynamics*, pages 193–207, 1994.

[LR04] Benedict Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Cambridge University Press, 2004.

[LR05] Ben Leimkuhler and Sebastian Reich. *Simulating Hamiltonian Dynamics*. Cambridge University Press, 2005.

[LSSP02] Stefan M. Larson, Christopher D. Snow, Michael R. Shirts, and Vijay S. Pande. Folding@Home and Genome@Home: Using distributed computing to tackle previously intractable problems in computational biology. *Computational Genomics*, 2002.

[LvdPF00] Joseph Laszlo, Michiel van de Panne, and Eugene L. Fiume. Interactive control for physically-based animation. In *Proc. of ACM SIGGRAPH 2000*, pages 201–208, July 2000.

[MAB$^+$97] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. K. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. Design Galleries: A general approach to setting parameters for computer graphics and animation. In *Proc. of ACM SIGGRAPH 1997*, pages 389–400, August 1997.

[Mau97] Michael I. Mauldin. Lycos: design choices in an Internet search service. *IEEE Expert*, 12(1):8–11, January 1997.

[MC95] Brian Mirtich and John Canny. Impulse-based simulation of rigid bodies. In *1995 Symposium on Interactive 3D Graphics*, pages 181–188, April 1995.

[Mir98] Brian Mirtich. V-Clip: Fast and robust polyhedral collision detection. *ACM Transactions on Graphics*, 17(3):177–208, July 1998.

[Mir00] Brian Mirtich. Timewarp rigid body simulation. In *Proc. of ACM SIGGRAPH 2000*, pages 193–200, July 2000.

[MMS04] Viorel Mihalef, Dimitris Metaxas, and Mark Sussman. Animation and control of breaking waves. In *Proc. 2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 315–324, July 2004.

[MS01] Victor J. Milenkovic and Harald Schmidl. Optimization-based animation. In *Proc. of ACM SIGGRAPH 2001*, pages 37–46, August 2001.

[MT92] Dimitri Metaxas and Demetri Terzopoulos. Dynamic deformation of solid primitives with constraints. In *Computer Graphics (Proc. of SIGGRAPH 92)*, pages 309–312, July 1992.

[MTPS04] Antoine McNamara, Adrien Treuille, Zoran Popović, and Jos Stam. Fluid control using the adjoint method. *ACM Trans. on Graphics*, 23(3):449–456, August 2004.

[NC93] G. Nave and A. Cohen. ECG compression using long-term prediction. *IEEE Trans. on Biomedical Engineering*, 40(9):877–885, 1993.

[NH98] R. Nygaard and D. Haugland. Compressing ECG signals by piecewise polynomial approximation. In *Proc. of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing*, volume 3, pages 1809–1812, May 1998.

[ODGK03] Carol O'Sullivan, John Dingliana, Thanh Giang, and Mary K. Kaiser. Evaluating the visual fidelity of physically based animations. *ACM Trans. on Graphics*, 22(3):527–536, July 2003.

[Ogg] Vorbis I specification. http://www.xiph.org/vorbis/doc/Vorbis_I_spec.html.

[OH99] James F. O'Brien and Jessica K. Hodgins. Graphical modeling and animation of brittle fracture. In *Proceedings of SIGGRAPH 99*, Computer Graphics Proceedings, Annual Conference Series, pages 137–146, August 1999.

[O'S05] Carol O'Sullivan. Collisions and attention. *ACM Trans. Appl. Percept.*, 2(3):309–321, y 05.

[PB61] William Wesley Peterson and D. T. Brown. Cyclic codes for error detection. *Proceedings of the IRE*, 49(1):228–235, January 1961.

[PB81] Stephen M. Platt and Norman I. Badler. Animating facial expressions. In *Computer Graphics (Proc. of SIGGRAPH 81)*, pages 245–252, 1981.

[PB88] John C. Platt and Alan H. Barr. Constraints methods for flexible models. In *Computer Graphics (Proc. of SIGGRAPH 88)*, pages 279–288, 1988.

[PBMW98] Lawrence Page, Sergey Brin, Rajeev Motwani, and Terry Winograd. The PageRank Citation Ranking: Bringing Order to the Web. Technical Report SIDL-WP-1999-0120, Stanford Digital Libraries, 1998.

[PC93] K. C. Park and J. C. Chiou. A discrete momentum-conserving explicit algorithm for rigid body dynamics analysis. *International Journal for Numerical Methods in Engineering*, 36(7):1071–1083, 1993.

[PCS04] Frédéric Pighin, Jonathan M. Cohen, and Maurya Shah. Modeling and editing flows using advected radial basis functions. In *Proc. 2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 223–232, July 2004.

[PF02] Kevin B. Pratt and Eugene Fink. Search for patterns in compressed time series. *International Journal of Image and Graphics*, 2(1):89–106, 2002.

[PSE$^+$00] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew P. Witkin. Interactive manipulation of rigid body simulations. In *Proc. of ACM SIGGRAPH 2000*, pages 209–218, July 2000.

[PSE03] Jovan Popović, Steven M. Seitz, and Michael Erdmann. Motion sketching for control of rigid-body simulations. *ACM Trans. on Graphics*, 22(4):1034–1054, October 2003.

[PV99] Paolo Prandoni and Martin Vetterli. Approximation and compression of piecewise smooth functions. *Phil. Trans. R. Soc. Lond. A*, 357(1760):2573–2591, 1999.

[RAP08] P. S. A. Reitsma, J. Andrews, and N. S. Pollard. Effect of character animacy and preparatory motion on perceptual magnitude of errors in ballistic motion. *Computer Graphics Forum*, 27(2):201–210, April 2008.

[Rei99] Hans Reichenbach. *The Direction of Time*. Dover Publications, 1999.

[REN$^+$04] Nick Rasmussen, Doug Enright, Duc Nguyen, S. Marino, Nigel Sumner, Willi Geiger, Samir Hoon, and Ron Fedkiw. Directable photorealistic liquids. In *Proc. 2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 193–202, July 2004.

[RNGF03] Nick Rasmussen, Duc Quang Nguyen, Willi Geiger, and Ronald P. Fedkiw. Smoke simulation for large-scale phenomena. *ACM Trans. on Graphics*, 22(3):703–707, July 2003.

[Rob94] Tony Robinson. SHORTEN: Simple lossless and near-lossless waveform compression. Technical Report CUED/D-INFENG/TR.156, Cambridge University Engineering Department, December 1994.

[RP03] Paul S. A. Reitsma and Nancy S. Pollard. Perceptual metrics for character animation: Sensitivity to errors in ballistic motion. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):537–542, July 2003.

[RPE$^+$05] Liu Ren, Alton Patrick, Alexei A. Efros, Jessica K. Hodgins, and James M. Rehg. A data-driven approach to quantifying natural human motion. *ACM Transactions on Graphics (SIGGRAPH 2005)*, 24(3), August 2005.

[Sal04] David Salomon. *Data Compression*. Springer-Verlag New York, Inc., third edition, 2004.

[SAM$^+$04] Anthony Sherbondy, David Akers, Rachel Mackenzie, Robert Dougherty, and Brian Wandell. Exploring connectivity of the brain's white matter with dynamic queries. In *Proc. of IEEE Visualization 2004*, pages 377–384, 2004.

[SDE05] Joshua Schpok, William T. Dwyer, and David S. Ebert. Modeling and animating gases with simulation features. In *Proc. 2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 97–105, July 2005.

[SF98] Karan Singh and Eugene L. Fiume. Wires: A geometric deformation technique. In *Proc. of ACM SIGGRAPH 1998*, pages 405–414, July 1998.

[SH96] D. Stoianovici and Y. Hurmuzlu. A critical study of the applicability of rigid-body collision theory. *Journal of Applied Mechanics*, 63:307–316, June 1996.

[SH07] Alla Safonova and Jessica K. Hodgins. Construction and optimal search of interpolated motion graphs. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), August 2007.

[SHC05] Julian Smart, Kevin Hock, and Stefan Csomor. *Cross-Platform GUI Programming with wxWidgets*. Prentice Hall PTR, 2005.

[Sho85] Ken Shoemake. Animating rotation with quaternion curves. In *Computer Graphics (Proc. of SIGGRAPH 85)*, pages 245–254, July 1985.

[Smi06] Russell Smith. *Open Dynamics Engine v0.5 Users Guide*, February 2006.

[SNF05] Eftychios Sifakis, Igor Neverov, and Ronald Fedkiw. Automatic determination of facial muscle activations from sparse motion capture marker data. *ACM Trans. on Graphics*, 24(3):417–425, August 2005.

[Spa91] Helmuth Spaeth. *Mathematical Algorithms for Linear Regression*. Academic Press, 1991.

[SRF05] Andrew Selle, Nick Rasmussen, and Ronald Fedkiw. Space battle pyromania. In *Proceedings of the SIGGRAPH 2005 Conference on Sketches & Applications*. ACM Press, August 2005.

[SSK05] Mirko Sattler, Ralf Sarlette, and Reinhard Klein. Simple and efficient compression of animation sequences. In *Proc. 2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 209–218, July 2005.

[ST96] D. E. Stewart and Jeff C. Trinkle. An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and Coulomb friction. *Intl. J. Num. Meth. Eng.*, 39:2673–2691, 1996.

[ST97] D. E. Stewart and Jeff C. Trinkle. Dynamics, friction, and complementarity problems. In M. C. Ferris and J. S. Pang, editors, *Complementarity and Variational Problems*, pages 425–439. SIAM, 1997.

[Ste98] D.E. Stewart. Convergence of a time-stepping scheme for rigid body dynamics and resolution of Painlevé's problems. *Archive Rational Mechanics and Analysis*, 145(3):215–260, 1998.

[Ste00] David E. Stewart. Rigid body dynamics with friction and impact. *SIAM Review*, 42(1):3–39, 2000.

[Sto07] Jon Stokes. Intel picks up gaming physics engine for forthcoming GPU product. `http://arstechnica.com/news.ars/post/20070917-intel-picks-up-gaming-physics-engine-for-\forthcoming-gpu-product.html`, September 2007.

[Str90] W. J. Stronge. Rigid body collisions with friction. *Proc. R. Soc. Lond. A*, 431, 1990.

[SW91] J. C. Simo and K. K. Wong. Unconditionally stable algorithms for rigid body dynamics that exactly preserve energy and momentum. *International Journal for Numerical Methods in Engineering*, 31(1):19–52, January 1991.

[SY02] Lin Shi and Yizhou Yu. Object modeling and animation with smoke. Technical Report UIUCDCS-R-2002-2262, University of Illinois at Urbana-Champaign, January 2002.

[SY05a] Lin Shi and Yizhou Yu. Controllable smoke animation with guiding objects. *ACM Trans. on Graphics*, 24(1):140–164, January 2005.

[SY05b] Lin Shi and Yizhou Yu. Taming liquids for rapidly changing targets. In *Proc. 2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 229–236, July 2005.

[TBHF03] J. Teran, S. Blemker, V. Ng Thow Hing, and R. Fedkiw. Finite volume methods for the simulation of skeletal muscle. In *Proc. 2003 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 68–74, August 2003.

[TGP07] Stephan Trojansky, Thomas Ganshorn, and Oliver Pilarski. 300's liquid battlefield: fluid simulation Spartan style. In *SIGGRAPH '07: ACM SIGGRAPH 2007 sketches*, page 95, New York, NY, USA, 2007. ACM.

[TKPR06] Nils Thüry, Richard Keiser, Mark Pauly, and Ulrich Rüde. Detail-preserving fluid control. In *Proc. 2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 7–12, September 2006.

[TMPS03] Adrien Treuille, Antoine McNamara, Zoran Popović, and Jos Stam. Keyframe control of smoke simulations. *ACM Trans. on Graphics*, 22(3):716–723, July 2003.

[TNM95] Diane Tang, J. Thomas Ngo, and Joe Marks. N-body spacetime constraints. *The Journal of Visualization and Computer Animation*, 6(3):143–154, July–September 1995.

[TT94] Xiaoyuan Tu and Demetri Terzopoulos. Artificial fishes: Physics, locomotion, perception, behavior. In *Proc. of ACM SIGGRAPH 1994*, pages 43–50, July 1994.

[TW88] Demetri Terzopoulos and Andrew Witkin. Physically based models with rigid and deformable components. *IEEE Computer Graphics & Applications*, 8(6):41–51, November 1988.

[Uhl91] J. K. Uhlmann. Satisfying general proximity/similarity queries with metric trees. *Information Processing Letters*, 40(4):175–179, November 1991.

[vABHL03] Luis von Ahn, Manuel Blum, Nicholas J. Hopper, and John Langford. CAPTCHA: Using hard AI problems for security. In *Proceedings of Eurocrypt*, pages 294–311, 2003.

[vLMvW96] Robert van Liere, Jurriaan D. Mulder, and Jarke J. van Wijk. Computational steering. In *High Performance Computing and Networking 1996*, April 1996.

[Was04] Larry Wasserman. *All of Statistics*. Springer-Verlag New York, Inc., 2004.

[WG97] Jane Wilhelms and Allen Van Gelder. Anatomically based modeling. In *Proc. of ACM SIGGRAPH 1997*, pages 173–180, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[Wit01] Andrew P. Witkin. Constrained dynamics. In *Physically Based Modeling: SIGGRAPH 2001 Course 25*, 2001.

[WK88] Andrew Witkin and Michael Kass. Spacetime constraints. In *Computer Graphics (Proc. of SIGGRAPH 88)*, pages 159–168, August 1988.

[WMT06] Chris Wojtan, Peter J. Mucha, and Greg Turk. Keyframe control of complex particle systems using the adjoint method. In *Proc. 2006 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, pages 15–23, September 2006.

[WTF06] Rachel Weinstein, Joseph Teran, and Ron Fedkiw. Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE Trans. on Visualization and Computer Graphics*, 12(3):365–374, May 2006.

[WV97] Robert A. Wannamaker and Edward R. Vrscay. Fractal wavelet compression of audio signals. *J. Audio Eng. Soc.*, 45(7–8):540–553, 1997.