Many-Worlds Browsing for Control of Multibody Dynamics

Christopher D. Twigg[*]
Carnegie Mellon University

Doug L. James[†]
Cornell University

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.

We introduce *Many-Worlds Browsing,* a method which circumvents these limits by exploiting the speed of multibody simulators to compute numerous example simulations in parallel (offline and online), and allow the user to browse and modify them interactively. We demonstrate intuitive interfaces through which the user can select among the examples and interactively adjust those parts of the scene that do not match his requirements. We show that using a combination of our techniques, unusual and interesting results can be generated for moderately sized scenes with under an hour of user time. Scalability is demonstrated by sampling much larger scenes using modest offline computations.

CR Categories: I.3.5 [COMPUTER GRAPHICS]: Computational Geometry and Object Modeling—Physically based modeling; I.6.8 [SIMULATION AND MODELING]: Types of Simulation—Animation

Keywords: interactive animation, rigid body dynamics, control, browsing, data-driven animation

1 Introduction

The use of physically based simulation has greatly simplified the creation of effects such as water, fire, and interacting solids. However, tuning these simulations to get the desired results can be a difficult and time-consuming process. A number of techniques have been developed that allow users to specify what the desired result should look like, and then solve for the simulation parameters that produce good approximations while maintaining physical realism. While these techniques have been successfully applied to fluid animation [Treuille et al. 2003; Shi and Yu 2002; Fattal and Lischinski 2004], very little progress has been made in the control of multibody dynamics since 2003. There is moreover no reason to believe that recent advances in controlling fluids can be transferred over to complicated multibody systems, since gradient-based techniques struggle with the bifurcations in the parameter space that result from rigid body collision handling. Gradient-based methods such as those developed by Popović and colleagues [2000; 2003] either resort to random sampling in these cases or rely on a good initial guess to ensure convergence.

On the other hand, great strides have been made in the simulation of multibody dynamics in recent years. In addition to the impressive results seen in several recent academic papers [Guendelman et al. 2003; Kaufman et al. 2005; Weinstein et al. 2006], rigid dynamics have become ubiquitous in video games. A number of fast, stable solvers are available both commercially (e.g., Havok, Ageia) and as

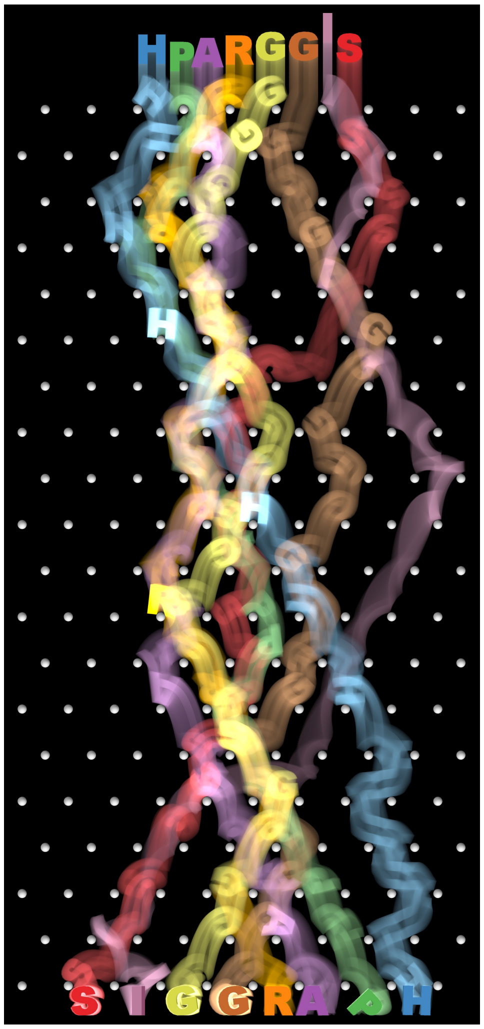


Figure 1: **Spelling SIGGRAPH:** *Using our parallel refinement with spatial queries and metrics enables a user to generate this animation spelling out "SIGGRAPH" from an arbitrarily chosen starting configuration.*

[*]e-mail: cdtwigg@cs.cmu.edu
[†]e-mail: djames@cs.cornell.edu

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.

This particular confluence of extremely fast simulation but slow continuous optimization would seem to suggest using a sampling-based approach to find animations that satisfy user constraints. This is especially true for larger scenes; having many collisions will cause gradient-based techniques to grind to a halt but can actually make sampling easier by increasing the variability in the data set. However, naïve sampling fails to find solutions to all but the most simple constraint problems. Chenney and Forsyth [2000] 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.

Even if we knew of some hypothetical optimization technique that was general enough to solve any arbitrary set of user constraints, there are still valid reasons for keeping the user in the loop. The specification of constraints precise enough to use in unattended optimization is itself time-consuming and error-prone in the absence of useful feedback. Since the sampling algorithm is only obligated to produce exactly what the user asked for, failure to specify implicitly understood constraints such as "look exciting" or "land right-side up" will produce undesired results. 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 substantially different approach. We will perform sampling in parallel on a cluster of machines to keep the entire approach interactive. We will give the user tools to steadily guide the sampling toward the result he seeks by telling the system which parts of the produced motion to keep and which to discard, and the system ensures that physical realism is maintained. We also describe how the data is compressed to increase the number of examples that can be stored locally on the user's machine.

To make this system feasible, 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 extremely successful searching interface made ubiquitous by Google, which takes the enormous amount of data present in the World Wide Web and makes it palatable. Google does this through two basic means: keywords, which constrain the set of pages that can be returned, and the Google PageRank metric [Page et al. 1998], which ranks the remaining results. 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.

We will show how these techniques can be used interactively to get specific results from scenes containing complicated articulated structures. We will also show that with the addition of offline computation, these methods can be scaled up to scenes involving hundreds of rigid bodies.

1.1 Related work

Control techniques were first applied to rigid bodies through actuated joints in the context of character animation, such as in Spacetime Optimization [Witkin and Kass 1988]. Tang and colleagues [1995] extended these techniques to passive simulation, using a genetic algorithm to search for the initial conditions that would produce the desired end goal, but only applied their algorithm to frictionless bodies in two dimensions. Barzel and colleagues [1996] introduced the idea of "plausibility," allowing additional control through the perturbation of collision normals. They also suggested a simple algorithm to search for simulations that satisfy constraints, but it is unlikely that it would scale far beyond the 2-dimensional pool ball simulations they present.

Chenney and Forsyth [2000] used the Markov Chain Monte Carlo algorithm (MCMC) to find solutions to pre-specified constraint problems. While their approach was successful in finding solutions to a variety of problems, it took many hours to run, which would make it impractical for many production situations. Furthermore, MCMC can be difficult to tune since good results depend on the Markov chain having rapid mixing [Wasserman 2004]; in the case of Chenney and Forsyth, this meant that the proposal mechanism which guided the sampling process was selected independently for each example. Popović and colleagues [2003] demonstrated a gradient-based method using multiple shooting that converged in 10-20 minutes for simulations involving several objects. It is not clear how well this technique would scale to simulations involving hundreds of interacting objects.

Popović and colleagues [2000] 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 [Baraff 1994].

A number of papers deal with the control of smoke and liquid animations involving thousands of degrees of freedom. These techniques generally rely on either the computation of simulation gradients [Treuille et al. 2003; McNamara et al. 2004] or the application of small, local forces at every time step [Shi and Yu 2002; Fattal and Lischinski 2004; Shi and Yu 2005a; Shi and Yu 2005b]. 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.

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 [Rasmussen et al. 2003; Mihalef et al. 2004]; however, these methods do not address the problem of how to assist the user in selecting which examples to use. Pighin and colleagues [2004] 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.

Our work on user interfaces for selecting among the computed examples is most closely related to Design Galleries [Marks et al. 1997]. 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 [2000] 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, since "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 [1992], 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 [2000] 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.

2 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 Section 5.

Plausibility: It is important that all the samples we generate be acceptable as answers to user queries, since 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 [2003] 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.

Most optimization techniques attempt to maintain plausibility by minimizing a scalar metric integrated over the entire simulation. For example, Chenney and colleagues [2000] define the *plausibility function* as the product of Gaussian distributions over surface normals during collision events, $p \propto \prod_i e^{-(\theta_i/10)^2/2}$ where θ_i is the angular displacement of the normal from its unmodified value. The advantage of such a metric is that it can prevent a sequence of "unlikely" collision events from occurring. However, we are aware of no evidence that a series of smaller perturbations will appear less likely than a single larger perturbation. Barzel and colleagues [1996] 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." If we select each impulse independently, though, this will happen with exponentially small probability. We therefore define plausibility as follows: *if each collision is individually plausible according to the criteria developed by O'Sullivan and colleagues [2003], 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 $\alpha = 0.1$.

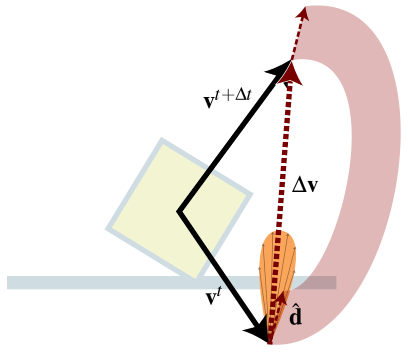


Figure 2: ***Applying Perturbations:*** *We apply a perturbation impulse $m\hat{\mathbf{d}}$ to the post-impact velocity $\mathbf{v}^{t+1}$ to produce the initial state for the next time step. The perturbation is sampled from a distribution about $\Delta\mathbf{v}$.*

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, $\mathsf{s}^t = \{\mathbf{x}_i^t, \Theta_i^t, \mathbf{v}_i^t, \omega_i^t\}$, 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 $\mathsf{s}^{t+\Delta t}$. 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

$$\Delta\mathbf{v}_i = (\mathbf{v}_i^{t+\Delta t} - \mathbf{v}_i^t)/\Delta t \tag{1}$$

and analogously, for angular velocity,

$$\Delta\omega_i = (\omega_i^{t+\Delta t} - \omega_i^t)/\Delta t \tag{2}$$

Suppose we have generated the simulation up to state s^t. We use our solver to generate the state $\mathsf{s}^{t+\Delta t}$. The result of our algorithm will be the perturbed state $\tilde{\mathsf{s}}^{t+\Delta t}$. 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 $\sin^n$ 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 2). We add $m\hat{\mathbf{d}}$ to $\mathbf{v}_i^{t+1}$ to produce the new post-impact state $\tilde{\mathsf{s}}^{t+\Delta t} = \{\mathbf{x}_i^{t+\Delta t}, \Theta_i^{t+\Delta t}, \mathbf{v}_i^{t+\Delta t} + m\hat{\mathbf{d}}, \omega_i^{t+\Delta t}\}$, 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.

Parallelization: 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.

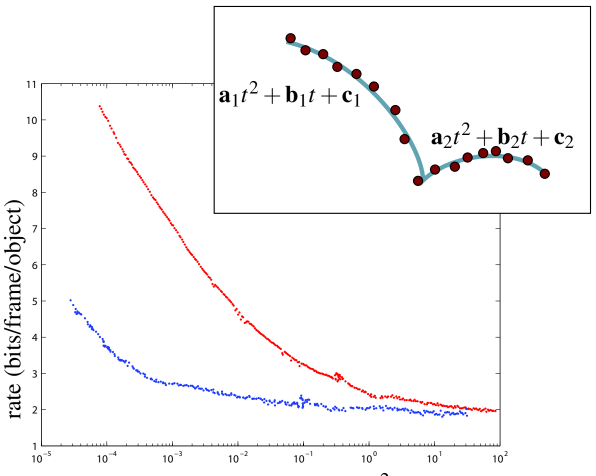


Figure 3: **Rate-distortion curve** *for the position component in our compression scheme; the blue curve corresponds to a collection of objects in free flight while the red curve corresponds to an articulated character. Notice that storage cost for the articulated character is significantly higher.* ***Inset:*** *For compression, we fit piecewise quadratic splines to object motions.*

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, with $\mathbf{a}, \mathbf{b}, \mathbf{c} \in \mathbb{R}^3$, as shown in Figure 3. 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 [Salomon 2004] 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, since 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, we only need to retain the quadratic coefficient for the vertical component of the object's position.

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

Figure 4: **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 $\mathscr{F} = [\mathbf{x}^{t_1}, \mathbf{x}^{t_2}, \ldots \mathbf{x}^{t_n}]$, 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.*

To maintain quality, we place a limit e_{max} on the ℓ^∞ norm of each segment. 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 $[t_1, t_2 \ldots t_n]$ we chose standard least squares over more expensive algorithms that explicitly minimize the ℓ^∞ norm. Selecting where to place transitions between splines is a harder problem. While it is possible to compute the optimal fit using dynamic programming [Nygaard and Haugland 1998; Haugland et al. 1997], these methods are expensive, and we use the much simpler greedy algorithm described in Figure 4 in our work.

Although it is possible to construct higher-order splines on rotation spaces using quaternions [Shoemake 1985], 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 [Grassia 1998], 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 [Kaufman et al. 2005]

$$[\omega] = \begin{pmatrix} 0 & -\omega_2 & \omega_1 \\ \omega_2 & 0 & -\omega_0 \\ -\omega_1 & \omega_0 & 0 \end{pmatrix} \tag{3}$$

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 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 $(\Theta_i^{t_i})^{-1} e^{[(t_i - t_0)\omega/(t_n - t_0)]}$.

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.

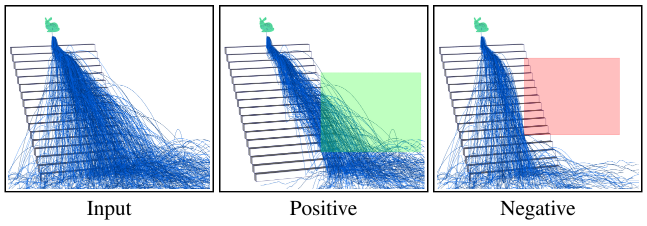


Figure 5: **Spatial queries** *in screen space are used to include and exclude simulation examples. (left) All input simulation examples for a single body example; (middle) Positive simulation constraint boxes (green) include simulation worlds with motion inside; (right) Negative simulation constraint boxes (red) exclude simulation worlds with motion inside.*

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 [2004]. 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 [2004] applied a similar paradigm in the context of diffusion tensor imaging, using queries to select among pathways in the brain.

Our queries will be similar but will operate on the center-of-mass trajectories of simulated objects, as shown in Figure 5. 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 [2000] 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.

4.2 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 [2000]. 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.

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

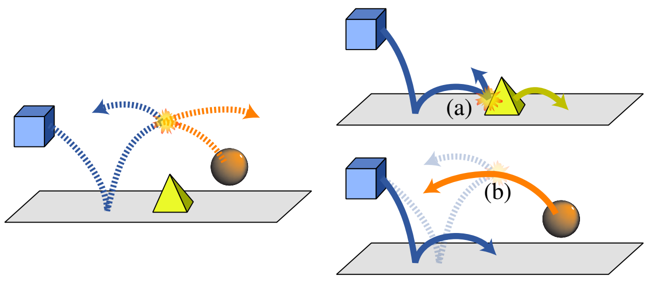


Figure 6: ***Refining objects:*** *If we refine the original (dotted) path taken by the blue box (left), we must ensure that other objects in the scene still have physically valid motion. Specifically, if the new motion interacts with other objects in the scene (as at point (a), right), we must simulate 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.*

```
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 $j \in \mathcal{O}_g$ : Symm. 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 7: ***Algorithm for 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 $\mathcal{O}_g$.*

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.

- **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.

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 $\mathsf{s}^{t_{min}}$. 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 6(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 6(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 6(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 6(b)); these must be added to $\mathcal{O}_a$ to guarantee physical validity.

The algorithm for updating these sets is shown in Figure 7. 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

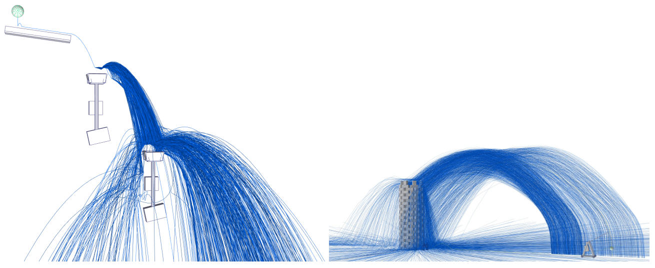


Figure 8: **Browsing trajectories:** *(left) In our* ***refrigerator toy*** *example, rotating buckets can be added even after sampling has begun; (right) after precomputation, over 1500 simulations with this* ***trebuchet and tower*** *can be loaded and browsed interactively.*

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.

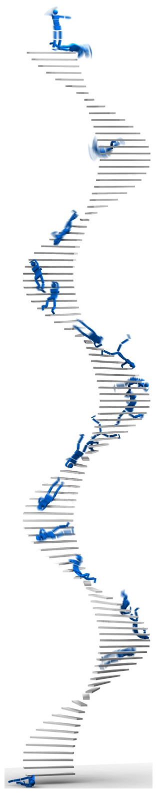
One useful extension to this technique can be developed by noting that there is nothing about the REFINE algorithm described above that necessitates that the sets $\mathcal{O}_a$ and $\mathcal{O}_g$ be in perfect correspondence. We could for example add a new object to the scene by simply inserting it 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 motion 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.

6 Results

We applied our technique to several examples. Our interactions with the system can be seen in the accompanying videos (longer versions are available on the project website listed in section 7). Dynamics were computed using the Open Dynamics Engine (ODE) [Smith 2006].

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 8). 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 in the desired direction. The user is able to build a particular chain of four rotating cups in just under five and a half minutes.

Spelling SIGGRAPH: The large number of collisions in this Pachinko-style environment (see Figure 1) 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. The user was able to quickly pick up on several strategies such as first handling the objects that had the farthest horizontal distance to travel and refining objects in pairs at a time. It would be possible to develop algorithms that used these same insights, but hard to develop an algorithm that could generalize as quickly as a person can.



Spiral staircase: The goal was to generate an animation where a simple articulated character model falls all the way down a three-level spiral staircase without falling off. 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.

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 8). 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."

7 Conclusion and Discussion

We have presented a simple interactive technique for extending the control of multibody dynamics to problems involving large numbers of collisions and articulated bodies. We believe that much of the power of our method lies in its simplicity, which will hopefully facilitate wider adoption, and its leveraging of human capabilities.

Our method has several limitations. First, more difficult problems require user involvement in the sampling process, which could be seen as a step back from previous methods that run unattended once user goals have been specified. Especially improbable scenes might require a great deal of user interaction. Furthermore, it is not clear how intuitive refinement may be for the inexperienced user. While the authors were able to quickly pick up how to use the system, more users (and/or user studies) are needed to establish how intuitive or useful Many-Worlds Browsing really is. To facilitate this exploration, we have made our application available at

`http://graphics.cs.cmu.edu/projects/mwb`

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. Finally, as simulators for phenomena such as cloth and hair become faster it may become possible to extend our work to these domains, but substantial work may be required to enable the storage of and interaction with such high-dimensional data.

Acknowledgments: The authors would like to thank the anonymous reviewers for their helpful comments; Chris Cameron for help rendering; James Kuffner for helpful discussions; and Matthias Müller, Emmanuel Marquez, and Philipp Hatt for assistance with the Ageia PhysX SDK. This work was supported in part by by a NVIDIA Graduate Fellowship, National Science Foundation grant CAREER-0430528, Pixar, the Alfred P. Sloan Foundation, The Boeing Company, hardware donations by NVIDIA and Intel, and Maya licenses donated by Autodesk.

References

ASSARSSON, U., AND MÖLLER, T. 2000. Optimized view frustum culling algorithms for bounding boxes. *Journal of Graphics Tools 5*, 1, 9–22.

BARAFF, D. 1994. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of ACM SIGGRAPH 1994*, 23–34.

BARZEL, R., HUGHES, J. F., AND WOOD, D. 1996. Plausible motion simulation for computer animation. In *EGCAS '96: Seventh International Workshop on Computer Animation and Simulation*.

CHENNEY, S., AND FORSYTH, D. A. 2000. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of ACM SIGGRAPH 2000*, 219–228.

COHEN, M. F. 1992. Interactive spacetime control for animation. In *Computer Graphics (Proceedings of SIGGRAPH 92)*, 293–302.

FATTAL, R., AND LISCHINSKI, D. 2004. Target-driven smoke animation. *ACM Transactions on Graphics 23*, 3 (Aug.), 441–448.

GRASSIA, F. S. 1998. Practical parameterization of rotations using the exponential map. *Journal of Graphics Tools 3*, 3, 29–48.

GUENDELMAN, E., BRIDSON, R., AND FEDKIW, R. P. 2003. Nonconvex rigid bodies with stacking. *ACM Transactions on Graphics 22*, 3 (July), 871–878.

HAUGLAND, D., HEBER, J. G., AND HUSØY, J. H. 1997. Optimisation algorithms for ECG data compression. *Medical and Biological Engineering and Computing 35*, 420–424.

HOCHHEISER, H., AND SHNEIDERMAN, B. 2004. Dynamic query tools for time series data sets: Timebox widgets for interactive exploration. *Information Visualization 3*, 1–18.

KAUFMAN, D. M., EDMUNDS, T., AND PAI, D. K. 2005. Fast frictional dynamics for rigid bodies. *ACM Transactions on Graphics 24*, 3 (Aug.), 946–956.

LASZLO, J., VAN DE PANNE, M., AND FIUME, E. L. 2000. Interactive control for physically-based animation. In *Proceedings of ACM SIGGRAPH 2000*, 201–208.

MARKS, J., ANDALMAN, B., BEARDSLEY, P. A., FREEMAN, W., GIBSON, S., HODGINS, J. K., KANG, T., MIRTICH, B., PFISTER, H., RUML, W., RYALL, K., SEIMS, J., AND SHIEBER, S. 1997. Design galleries: A general approach to setting parameters for computer graphics and animation. In *Proceedings of ACM SIGGRAPH 1997*, 389–400.

MCNAMARA, A., TREUILLE, A., POPOVIĆ, Z., AND STAM, J. 2004. Fluid control using the adjoint method. *ACM Transactions on Graphics 23*, 3 (Aug.), 449–456.

MIHALEF, V., METAXAS, D., AND SUSSMAN, M. 2004. Animation and control of breaking waves. In *Proceedings of the 2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 315–324.

NYGAARD, R., AND HAUGLAND, D. 1998. Compressing ECG signals by piecewise polynomial approximation. In *Proceedings of the 1998 IEEE International Conference on Acoustics, Speech, and Signal Processing*, vol. 3, 1809–1812.

O'SULLIVAN, C., DINGLIANA, J., GIANG, T., AND KAISER, M. K. 2003. Evaluating the visual fidelity of physically based animations. *ACM Transactions on Graphics 22*, 3 (July), 527–536.

PAGE, L., BRIN, S., MOTWANI, R., AND WINOGRAD, T. 1998. The PageRank Citation Ranking: Bringing Order to the Web. Tech. Rep. SIDL-WP-1999-0120, Stanford Digital Libraries.

PIGHIN, F., COHEN, J. M., AND SHAH, M. 2004. Modeling and editing flows using advected radial basis functions. In *Proceedings of the 2004 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 223–232.

POPOVIĆ, J., SEITZ, S. M., ERDMANN, M., POPOVIĆ, Z., AND WITKIN, A. P. 2000. Interactive manipulation of rigid body simulations. In *Proceedings of ACM SIGGRAPH 2000*, 209–218.

POPOVIĆ, J., SEITZ, S. M., AND ERDMANN, M. 2003. Motion sketching for control of rigid-body simulations. *ACM Transactions on Graphics 22*, 4 (Oct.), 1034–1054.

RASMUSSEN, N., NGUYEN, D. Q., GEIGER, W., AND FEDKIW, R. P. 2003. Smoke simulation for large-scale phenomena. *ACM Transactions on Graphics 22*, 3 (July), 703–707.

SALOMON, D. 2004. *Data Compression*, third ed. Springer-Verlag New York, Inc.

SHERBONDY, A., AKERS, D., MACKENZIE, R., DOUGHERTY, R., AND WANDELL, B. 2004. Exploring connectivity of the brain's white matter with dynamic queries. In *Proceedings of IEEE Visualization 2004*, 377–384.

SHI, L., AND YU, Y. 2002. Object modeling and animation with smoke. Tech. Rep. UIUCDCS-R-2002-2262, University of Illinois at Urbana-Champaign, Jan.

SHI, L., AND YU, Y. 2005. Controllable smoke animation with guiding objects. *ACM Transactions on Graphics 24*, 1 (Jan.), 140–164.

SHI, L., AND YU, Y. 2005. Taming liquids for rapidly changing targets. In *Proceedings of the 2005 ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, 229–236.

SHOEMAKE, K. 1985. Animating rotation with quaternion curves. In *Computer Graphics (Proceedings of SIGGRAPH 85)*, 245–254.

SMITH, R. 2006. *Open Dynamics Engine v0.5 Users Guide*, Feb.

TANG, D., NGO, J. T., AND MARKS, J. 1995. *N*-body spacetime constraints. *The Journal of Visualization and Computer Animation 6*, 3 (July–Sept.), 143–154.

TREUILLE, A., MCNAMARA, A., POPOVIĆ, Z., AND STAM, J. 2003. Keyframe control of smoke simulations. *ACM Transactions on Graphics 22*, 3 (July), 716–723.

WASSERMAN, L. 2004. *All of Statistics*. Springer-Verlag New York, Inc.

WEINSTEIN, R., TERAN, J., AND FEDKIW, R. 2006. Dynamic simulation of articulated rigid bodies with contact and collision. *IEEE Transactions on Visualization and Computer Graphics 12*, 3 (May), 365–374.

WITKIN, A., AND KASS, M. 1988. Spacetime constraints. In *Computer Graphics (Proceedings of SIGGRAPH 88)*, 159–168.