Sampling Plausible Solutions to Multi-body Constraint Problems

Stephen Chenney   D. A. Forsyth*

University of California at Berkeley

Abstract

Traditional collision intensive multi-body simulations are difficult to control due to extreme sensitivity to initial conditions or model parameters. Furthermore, there may be multiple ways to achieve any one goal, and it may be difficult to codify a user's preferences before they have seen the available solutions. In this paper we extend simulation models to include plausible sources of uncertainty, and then use a Markov chain Monte Carlo algorithm to sample multiple animations that satisfy constraints. A user can choose the animation they prefer, or applications can take direct advantage of the multiple solutions. Our technique is applicable when a probability can be attached to each animation, with "good" animations having high probability, and for such cases we provide a definition of physical plausibility for animations. We demonstrate our approach with examples of multi-body rigid-body simulations that satisfy constraints of various kinds, for each case presenting animations that are true to a physical model, are significantly different from each other, and yet still satisfy the constraints.

CR Descriptors: I.3.7 [**Computer Graphics**]: Three-Dimensional Graphics and Realism - *Animation*; I.3.5 [**Computer Graphics**]: Computational Geometry and Object Modeling - *Physically based modeling*; I.6.5 [**Simulation and Modeling**]: Model Development - *Modeling methodologies* G.3 [**Probability and Statistics**]: Probabilistic algorithms;

Keywords: plausible motion, Markov chain Monte Carlo, motion synthesis, spacetime constraints

1 INTRODUCTION

Collision intensive multi-body simulations are difficult to constrain because they exhibit extreme sensitivity to initial conditions or other simulation parameters. Adding uncertainty to a model helps when looking for animations that satisfy constraints [3], because it adds physically motivated degrees of freedom in useful places. For example, we can control tumbling dice by placing random bumps in specific places on the table, rather than by adjusting the initial conditions of the throw. The bumps are more effective because a small change to a bump part-way through the animation has a limited effect on where the dice land, but a small change in the initial conditions generally has an unpredictable effect. It is difficult to design efficient control algorithms for the latter case.

*email: {schenney,daf}@cs.berkeley.edu

As discussed by Barzel, Hughes and Wood [3], adding randomness to a simulation gives additional benefits:

- The real world contains fine scale variation that traditional simulation models generally ignore. We can use randomness to model this variation by, for instance, replacing a perfectly flat surface with one speckled with random bumps (the same random bumps used for control above). Animations generated with the new model can more accurately reflect the behavior of the world. In training environments, this results in the subject developing skills more compatible with the real world: a driver trained on simulations of bumpy roads will be better prepared for real world road surfaces.
- Visually, procedural animations can be more believable when uncertainty is added. Without uncertainty, a perfectly round ball dropped vertically onto a perfectly flat table moves strangely, a situation that may be improved by slightly perturbing the collisions to make the ball deviate from the vertical.

In a world with uncertainty, we generally expect a constrained problem to have multiple solutions. It is difficult to know beforehand what solutions are available, which compounds any difficulties a user may have in codifying their preferences. Hence, it is perverse to use a solution strategy that seeks a single answer, rather, we prefer a technique that produces many solutions that reflect the range of possible outcomes. While for feature animation a user is expected to choose the one animation they prefer, other applications benefit directly from multiple solutions:

- Computer game designers can use different animations each time a game is played, making it less predictable and potentially more entertaining.
- Training environments can present trainees with multiple physically consistent scenarios that reflect the physics and variety of the real world.

We generate multiple animations that satisfy constraints by applying a Markov chain Monte Carlo (MCMC) algorithm to sample from a randomized model. A user supplies the model of the world, including the sources of uncertainty and the simulator that will generate an animation in the world. The user also supplies a function that gives higher values for "good" animations — those that are likely in the world and satisfy the constraints. Finally, a user must provide a means of proposing a new animation given an existing one. The algorithm we describe in this paper generates an arbitrarily long sequence of animations in which "good" animations are likely to appear.

In this paper, along with the algorithm, we describe the sorts of models we use and how we sample from them, discussing examples from the domain of collision intensive rigid-body simulation. No previous algorithm has been shown for the range and complexity of the multi-body simulations we present.

2 RELATED WORK

The idea of *plausible motion simulation*, including the exploitation of randomness to satisfy constraints, was introduced by Barzel, Hughes and Wood [3]. They show solutions to constrained problems where, for instance, a billiard ball is controlled by randomly varying the collision normal each time it hits a rail. We extend

their work by introducing the idea of sampling (instead of searching), giving a precise definition of plausibility, and by demonstrating MCMC's effectiveness on a wide range of difficult examples.

Motion synthesis algorithms aim to achieve a goal by finding an optimal set of control parameters and (sometimes) initial conditions. The goals described in the literature include finding good locomotion parameters [1, 8, 14, 16, 23, 26] and finding trajectories that satisfy constraints [2, 5, 9, 13, 15, 20, 32]. Some techniques [2, 5, 9, 13, 15, 20, 32] exploit explicit gradient information, but fail if the problem is too large (Popović discusses ways to reduce the problem size [25]) or the constraints are highly sensitive to, or discontinuous in, the control parameters. Randomized algorithms, such as simulated annealing [14, 16] (not a panacea [10, 11]), stochastic hill climbing [8], or evolutionary computing [1, 23, 26, 29], do not require gradients and may be suitable for collision intensive systems — Tang, Ngo and Marks [29] describe an example. Most of these methods return a single "best" animation, and hence may ignore other equally good, or even preferable solutions. The evolutionary computing solutions can exhibit variations within a population, which Auslander et. al. [1] refer to as different styles, but the number of examples is limited by the population size.

Multi-body constraint problems are good candidates for a *Design Galleries* [21] interface, in which a user browses through sample solutions to locate the one they prefer. Our work addresses the sampling aspect of a Design Galleries interface for multi-body constrained animations, but we do not consider other aspects of the interface.

3 ANIMATION DISTRIBUTIONS

The MCMC algorithm distinguishes itself from motion synthesis approaches by generating multiple, different, "good" animations that satisfy a set of constraints, but no "best" animation. To generate multiple plausible constrained animations, we must provide a model of the world defining:

- The objects in the world and their properties, including the sources of uncertainty.
- The simulator for generating animations in the world.
- The constraints to be satisfied by the animations.

For example, in a 2D animation of a ball bouncing on the table, we might have uncertainty in the normal vectors at the collision points, a constraint on the resting place of the ball, and a simulator that determines what happens when a 2D ball bounces on a table with arbitrary surface normals. We will use this example, from [3], throughout the next two sections.

A simulator used with our approach need not be physically accurate, or even physically based. Our 2D ball simulator is obviously non-physical, and the simulator we use in other examples has some problems with complex frictional behavior (section 5.2.3). In any case, we assume that if the simulator is given a plausible world as input it will produce a plausible animation, according to some definition of plausibility (see section 3.3).

3.1 Incorporating Uncertainty

We define a function, $p_w(A)$, representing the probability of any possible animation A that might arise in the world model. Intuitively, $p_w(A)$ should be large for animations that are likely in the world, and low for unlikely animations. For the 2D ball example, $p_{w,ball}(A)$ should be high if all the normal vectors used to generate the animation were close to vertical, and low if most of them were far from vertical. Let us further insist that $p_w(A)$ be non-negative and have finite integral over the domain implied by the random variables in the model, so that we can view $p_w(A)$ as an unnormalized probability density function defined on the space of animations.

Expanding on the 2D ball example, let us describe the direction of the normal vector for each collision i as an independent random variable, θ_i, distributed according to the (bell-shaped) Gaussian distribution with standard deviation of, say, 10.0 degrees. In that case we get:

$$p_{w,ball}(A) \propto \prod_i e^{-\frac{1}{2}\left(\frac{\theta_i}{10.0}\right)^2}$$

which is the product of density functions for each collision normal. Note that we are ignoring normalization constants, an omission we justify in section 4. Also, we could in principle measure a real table to infer the true distribution of surface normals, and use that instead.

3.2 Constraints

If we restrict our attention to animations that satisfy constraints, we are concerned with the distribution function $p_w(A|C)$, which is the conditional distribution of A given that it satisfies the constraints C. For the 2D ball example, if we want the ball to land in a particular place, we could generate samples from $p_{w,ball}(A|C)$ using an inverse approach: join the ball's start point to its end point using a sequence of parabolic hops and then infer which normal vectors were required to generate such a trajectory. However, using this approach we cannot directly ensure that the animation we generate is likely in the world, because it is difficult to know which hops to use to get a set of likely normal vectors.

Unfortunately, it is frequently impractical to sample directly from $p_w(A|C)$, because there is no way to find, without considerable effort, any reasonable animation in which the constraints are satisfied. For example, in multi-body simulations a forward simulation approach doesn't work because no published algorithm can directly specify a set of control parameters leading to satisfaction of multi-body constraints, without doing some form of iterative, expensive search. The inverse approach also looks intractable: it is not clear how to set trajectories for all the participants such that, for instance, objects do not pass through each other.

In such cases (like all the examples in this paper), we expand $p_w(A)$ to include a term for the constraints, resulting in a function $p(A)$. The new intuition is that $p(A)$ will be large for animations that are likely in the world *and* satisfy the constraints, and small for animations that are either implausible in the world or don't satisfy the constraints. We will refer to $p(A)$ as the *probability* of an animation. Note that now even animations that don't satisfy the constraints have non-zero probability, so if we sample from $p(A)$ we may get an animation that doesn't satisfy the constraints, which we must discard.

For the examples in this paper, we define:

$$p(A) \propto p_w(A)p_c(A)$$

where $p_c(A)$ depends only on how well the animation satisfies the constraints. If we want our 2D ball to land at a point whose distance, d, from the origin is small, we can define

$$p_{c,ball}(A) \propto e^{-\frac{1}{2}\left(\frac{d}{\sigma_d}\right)^2}$$

which is the Gaussian density function with standard deviation σ_d, which we discuss in section 5.1. This function gives higher values for distances near zero, and lower values as distances increase. Hence, for the 2D ball example:

$$p_{ball}(A) \propto e^{-\frac{1}{2}\left(\frac{d}{\sigma_d}\right)^2} \prod_i e^{-\frac{1}{2}\left(\frac{\theta_i}{10.0}\right)^2}$$

This paper describes a technique for generating animations such that those with high probability will appear more frequently than those with lower probability, but even some low probability events will occur — as in the real world, unlikely things sometimes happen. In other words, we will sample according to the distribution defined by $p(A)$.

3.3 What does "Plausible" mean?

The restrictions on $p(A)$ are quite weak, so we can describe many types of uncertainty and a wide variety of constraints. By phrasing the problem as one involving probabilities, we can leverage a wide range of mathematical tools for talking about plausible motion, and make strong statements about the properties of the animations we generate (see section 4). We can also outline what it means to be physically plausible:

> A model, including its simulator, is plausible if the important statistics gathered from samples distributed according to $p(A)$ are sufficiently close to the real world statistics we care about.

This is a very general definition of plausibility, because we say nothing about which statistics we might care about, or what it means to be sufficiently close. For example, to validate a pool table model we could run simulations of virtual balls on a table, and analyze video of real balls on a real table, then compare statistics such as how long a ball rolls before coming to rest. For entertainment applications, we would care less about the quality of the match than if we were trying to build a training simulator for budding young pool sharks.

Our measure extends the traditional graphics idea of plausibility — "if it looks right it is right" — by allowing for definitions of statistical similarity other than a user's ability to detect a fake. However, for many applications, particularly involving motion, a viewer's ability to distinguish real from artificial remains the primary concern [17].

4 MCMC FOR ANIMATIONS

We use the Markov chain Monte Carlo (MCMC) method [12, 19] to sample animations from the distribution defined by $p(A)$. MCMC has several advantages for this task:

- MCMC generates a sequence, or *chain*, of samples, $A_0, A_1, A_2, \ldots$, that are distributed according to a given distribution, in this case $p(A)$.
- Apart from the initial sample, each sample is derived from the previous sample, which allows the algorithm to find and move among animations that satisfy constraints.
- If available, domain specific information can be incorporated into the algorithm, making it more efficient for special cases. On the other hand, the algorithm does not rely on any specific features of a model or simulator, allowing its application in a variety of situations.

Our MCMC algorithm for generating animations begins with an initial animation then repeatedly proposes changes, which may be accepted or rejected. Explicitly:

```
1      initialize(A_0)
2      simulate(A_0)
3      repeat
4            propose(A_c, A_i)
5            simulate(A_c)
6            u ← random(0, 1)
7            if u < min (1, p(A_c)q(A_i|A_c) / p(A_i)q(A_c|A_i))
8                  A_{i+1} ← A_c
9            else
10                 A_{i+1} ← A_i
```

Line 1 gives initial values to all the random variables in the world model. On line 4, a new animation, A_c, is proposed by making a random change to the previous animation, A_i. The details of this change are application specific. For example, in the 2D ball model of section 3 it might involve, for each normal, choosing to change it with probability one half and, if it is to be changed, adding a random offset uniformly distributed on $(-5, 5)$ degrees (for reasons

discussed in section 5.1). The probability of making changes is defined by the *transition probability*, $q(X|Y)$, which is the probability of proposing animation X if the current animation is Y. For the 2D ball, the transition probability is:

$$q_{ball}(X|Y) \propto \left(\frac{1}{2}\right)^n \cdot \left(\frac{1}{5 - (-5)}\right)^k$$

where n is the total number of collisions (assumed fixed) and k is the number of collisions that were changed. The first factor is the probability of choosing the particular set of normals to change, and the second factor codes the probability of choosing a particular offset for each normal that is changed.

The transition probabilities, along with the probabilities of the animations, are used in computing the *acceptance probability*, which is the probability of accepting the proposed candidate (line 7):

$$P_{accept} = \min\left(1, \frac{p(A_c)q(A_i|A_c)}{p(A_i)q(A_c|A_i)}\right)$$

Often, as in the 2D ball example, the transition probabilities are symmetric — $q(X|Y) = q(Y|X)$ — and will cancel. Note also that only the ratios of probabilities appear, so we can use functions that are only proportional to true probability density functions (section 3.1).

The proposal mechanism is one of the key factors in how well the algorithm will perform in a particular application. In practice, proposals are designed through intuitive reasoning and experimentation, using past experience as a guide. In section 5 we describe the motivation for our proposal mechanisms.

The MCMC algorithm guarantees that the samples in the chain will be distributed according to $p(A)$, as the number of samples approaches infinity and provided certain technical conditions are met [12]. Hence we can be certain that the samples our algorithm generates truly reflect the underlying model, and if this model is plausible (section 3.3), the collection of samples will be plausible. It is also the case that the samples in the chain will never satisfy the constraints if the underlying model says they cannot be satisfied. For instance, if a bowling simulator cannot capture complex frictional effects, animations that bowl the seven-ten split can never be found (see section 5.2.3).

MCMC has been used in graphics to generate fractal terrain that satisfies point constraints [28, 31]. The samples generated by an MCMC algorithm may also be used to estimate expectations, as in Veach's Metropolis algorithm for computing global illumination solutions [30]. In this paper we are not concerned with expectations, so we can use short chains, just long enough to satisfy a user with several different animations

5 EXAMPLES

We are interested in four things when designing an MCMC algorithm for generating animations:

- Is the motion plausible? We assume that the simulator produces plausible motion, so we are left to ensure that the distributions we use for the model are reasonable.
- How long does it take to find a sample that satisfies the constraints?
- How rapidly does the chain move among significantly different samples, or *mix*? Chains that mix faster are desirable because they produce many different animations quickly.
- How many of the samples satisfy the constraints well enough to be useful?

The following examples discuss issues in building models, defining constraints and selecting proposal strategies, all of which influence the behavior of the algorithm.

5.1 A 2D Ball

In the 2D ball example of section 3 a ball bounces on a table, starting in a fixed location and undergoing, for simplicity, a fixed number of collisions. For each collision we specify a random normal vector. The aim is to sample these normal vectors such that the ball comes to rest close to a particular location. As a specific case, we will drop the ball from above the origin at a height of $4.5D$, where D is the diameter of the ball, use five collisions, and specify that it come to rest near $x = D$ on the sixth collision.

The simulation model is: the ball moves ballistically between each collision, when the velocity of the ball is reflected about the corresponding normal vector and the normal component of velocity is scaled by $\frac{1}{\sqrt{2}}$. This model is not physically plausible (for instance, we are ignoring rotation effects), but for this example we value simplicity.

5.1.1 Uncertainty and Constraints

The probability of an animation is described in section 3.1, but probabilities (the values of density functions) can be very large numbers, so in practice we work with their logarithm. In this case, with x the horizontal position of the sixth collision:

$$\log(p(A)) = -\frac{1}{2}\left(\frac{x-D}{\sigma_d}\right)^2 - \frac{1}{2}\sum_{1 \leq i \leq 5}\left(\frac{\theta_i}{10.0}\right)^2 + C$$

for some constant C, which will cancel out when computing the acceptance probability.

The value of the constraint standard deviation, σ_d, has a major effect on the samples generated by the chain. Say we choose a small value for σ_d, corresponding to a very tight constraint because only values of x very close to D give high values for $p(A)$ and all other landing points have very low probability. From the initial animation, the chain will move to some high probability animation close to the constraint. But, once there, almost no new proposals are accepted (most candidates will be far from the constraint and have very low probability) and the user sees few different animations — an undesirable situation.

Alternatively, say we choose a large value for the standard deviation, corresponding to a weak constraint. Then $p(A)$ is relatively high for a wide range of landing positions. The result is undesirable: the chain will contain many high probability animations that are far from the constraints.

Hence we must choose a value for σ_d that is high enough to promote different samples but low enough to enforce the constraint. In this example we use a value of $0.1D$, where D is the diameter of the ball, which, as figure 2 shows, leads to the generation of very different samples that generally are close to the constraint. In this case, the algorithm is not very sensitive to the exact value for σ_d (anything within a factor of five works fine) and it is possible to experimentally evaluate a few values on short chains and choose the best, which in this case took only a few minutes.

In other applications there is no guarantee that we can achieve both good constraints and good mixing. In such cases the algorithm must run for many iterations to generate different samples, which may take prohibitively long. The tumbling dice example of section 5.4 is a borderline example in which we can satisfy constraints but mixing is poor. In such cases it is possible to run multiple chains in parallel.

5.1.2 Proposals

The proposal mechanism, which specifies normal vectors for a candidate animation, A_c, given those for the current animation, A_i, provides a means of moving around the space of possible normal vectors:

$$\textbf{for } j = 1 \textbf{ to } 5$$

$$A_c.normal\,[j] \leftarrow A_i.normal\,[j]$$
$$\textbf{if } \text{random}(0,1) < 0.5$$
$$\quad A_c.normal\,[j] \leftarrow A_c.normal\,[j] + \text{random}(-5,5)$$

This proposal changes some of the normals by an amount between minus one half and half their standard deviation of 10.0 degrees. For good mixing it is important to allow more than one normal to be changed at once, because the effect of each change on the landing position (and hence the constraint) can then cancel. The alternative, changing only one normal, makes it very difficult to change the first collision normal, because any but the smallest change will move the ball far from the desired landing position, and hence be rejected. The size of the offset we add is chosen to allow both small changes and relatively large changes, but not so large as to shift the normals too far from their mean in one step, which would reduce their probabilities and result in rejection of the candidate animation.
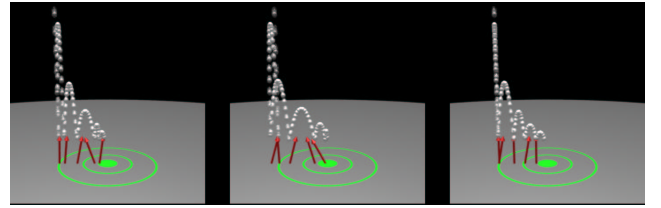
5.1.3 An Example Chain

Figure 1: *Three sample paths from the 2D ball example, plotting the trajectory of the center of the ball (although the plot is 3D, the ball moves only in 2D). The green target is centered on the constraint. Each red arrow is located at a collision point and indicates the direction of the normal vector used at that point. Note that in each example one of the earlier normals pushes the ball toward the constraint, and later normals refine the final position. One ball bounces slightly away from the constraint before moving toward it, which is not implausible.*

We ran the MCMC algorithm and generated a chain containing one thousand samples (many of these are repeats, arising when a candidate is rejected). Figure 2 plots the horizontal resting position of each sample. The first sample was initialized with randomly chosen normals, and came to rest a long way from the constraint. But within twenty iterations the chain moved toward a good location. The bumpiness of the graph indicates good mixing, because flat spots would indicate many repetitions of one sample as candidates were rejected. The majority of animations have the ball coming to rest within $0.1D$ of the desired position, indicating that σ_d is sufficiently small to enforce the constraint.

Three (randomly chosen) samples from the chain are shown in figure 1. They do not differ greatly from what one would expect: the ball tends to take an early bounce toward the constraint and keep moving in that direction, with later collisions adjusting it's final position.

5.2 Bowling

In this scenario the aim is to animate any particular ten-pin bowling shot (a goal suggested by Tang, Ngo and Marks [29]). The physical model is implemented by an impulse-based rigid-body simulator [6]. We model the bowling ball, the lane with simplified gutters and side walls, and the pins. All the models are roughly based on the rules of bowling, including variations allowed by those rules (see appendix A.1 for details):

- The ball is simulated as a sphere, with variable radius, density, initial position, initial velocity and initial angular velocity.

Strike $\longrightarrow$

Six-seven Split $\longrightarrow$
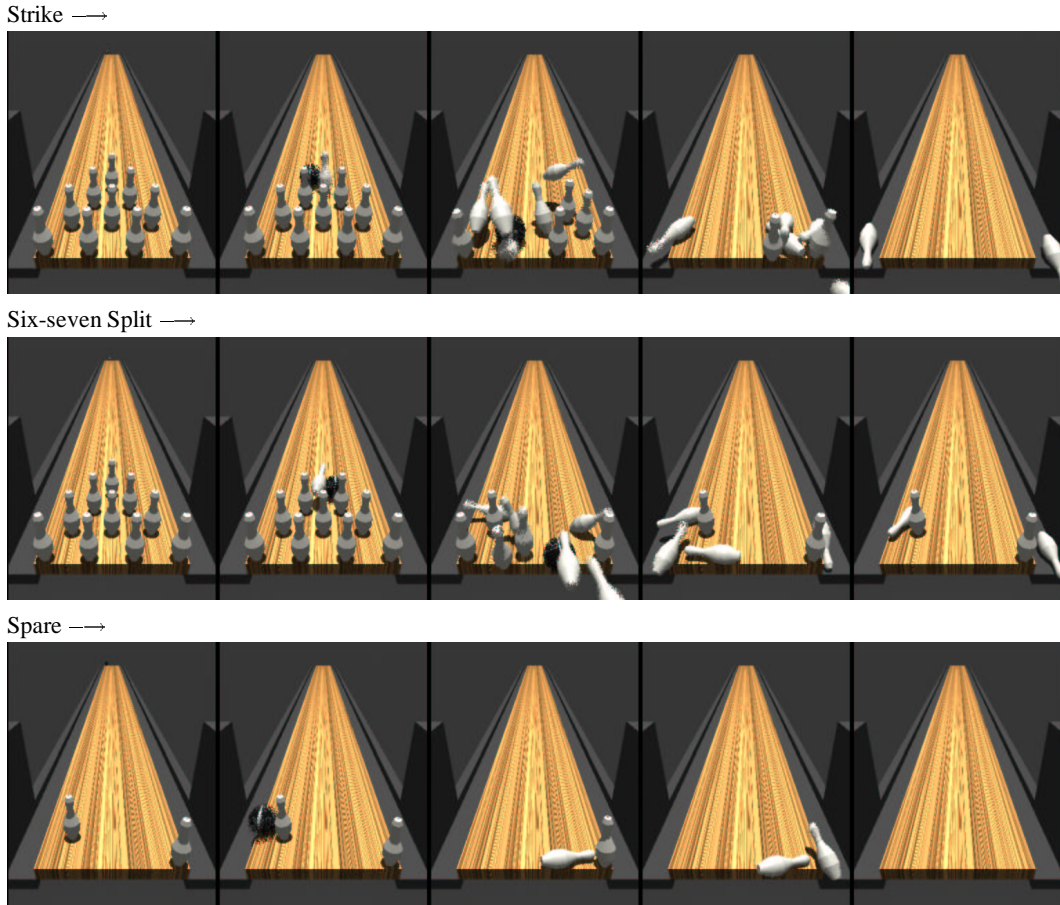

Spare $\longrightarrow$

Figure 3: *Frames from three bowling examples. The initial conditions for the ball and the pin locations are random variables. Given an initial and final pin configuration, the MCMC algorithm samples particular values for the random variables that lead to the desired shot. In this case, we demanded a strike, a six-seven split and the corresponding spare.*

- The lane is fixed with regulation length and width, and includes rectangular gutters and side walls starting in line with the front pin.
- Each pin, of fixed shape and mass, has its initial position on the lane perturbed by a small random amount.

The coefficients of friction and restitution between all the components are fixed. The probability $p_w(A)$ is proportional to the product of the distribution functions for each of the random variables in the model.

5.2.1 Constraints

The simulation begins with a subset of pins specified by the user, so we can specify the initial conditions for bowling spares. The user also sets the constraint by stating which pins should be knocked down and which should remain standing. We are unable to propose candidates for the MCMC algorithm that are certain to satisfy the constraints (section 3.2), so we assign non-zero probability to every possible outcome, but assign higher probability to those outcomes that are closer to the target, and the highest probability to outcomes matching the target. This is achieved with the Gibbs distribution function:

$$p_c(A) \propto \lambda^{k+m}$$

for some constant $\lambda > 1$ with k the number of pins that end up correctly standing or knocked down, and m the number of standing pins that have not moved far beyond their initial position. Animations that do not meet the goals will sometimes appear in the chain

(they have non-zero probability), but these would not be shown to a user. The samples that remain are correctly distributed according to the conditional probability $p(A|C)$, the distribution of animations in which the constraints are fully satisfied. The constraint involves a term derived from the pins' final position because some simulations result in the pins being pushed but not knocked down — behavior we wish to discourage.

The value of λ affects the proportion of animations in the chain that must be discarded for not satisfying the constraints. High values for λ give animations satisfying the constraints much higher probability, making them more likely to appear in the chain. But the chain mixes better if some "bad" animations appear. Say only perfect animations appear, then getting to a significantly different animation requires making a big change that also happens to get all the pins correct, which is unlikely. If some pins are not correct, a big change only has to get the same number of pins correct, and they can be different pins. A low value for λ makes it easier to accept an animation with some incorrect pins, make big changes, and then move toward a different, fully correct state.

For this example, we used $\lambda = e^{2.5}$, which gives a wide variety of animations that satisfy the constraints. Animations that improve the constraints are favored enough to ensure that good animations come up often, but not so much as to inhibit mixing.

Our use of the Gibbs distribution was motivated by other applications of the MCMC algorithm, such as counting the number of perfect matchings in a graph. It is known [18] that there is an optimal λ that balances the concerns outlined above, but that the algorithm
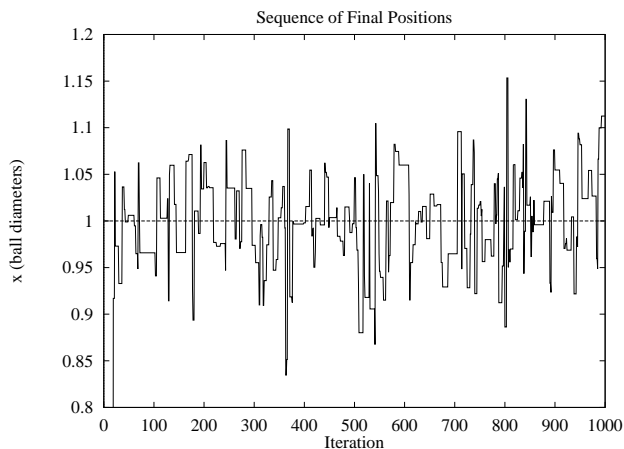
Figure 2: *The resting position of the first one thousand samples in a chain for the 2D ball example. The roughness of this graph indicates good mixing, and most samples are close to the constraint (the majority within $0.1D$). The position of the first few samples are far from the constraint (off the graph), but the chain moves to samples within twenty iterations.*

is relatively insensitive to its exact value. Experience suggests that many applications may exhibit similar behavior [27]: there exists a range of values for λ that give the chain good properties, and one such value may be found through experiment. Our results are consistent with this (also see section 5.3).

5.2.2 Proposals

Our proposal mechanism for bowling randomly chooses to do one of several things:

- Sample new values for all the random variables.
- Change the radius, density or initial conditions of the ball.
- Change the initial position of some pins.

The details are given in appendix A.2.

The first proposal strategy, which changes every random variable in the simulation, serves to make very large changes in the simulation. These are desirable as a means of escaping low probability regions, which we discuss in more detail in the next example (section 5.3). The other transitions are based on ideas similar to those in section 5.1: we must move around among possible values for the random variables, and we wish to do so with both large and small steps, but not so large as to make the new value highly unlikely under the model.

5.2.3 Sample Animations

We tested this model with three sets of constraints:

- Bowl a strike.
- Bowl a ball that leaves a six-seven split.
- Bowl the spare that knocks down the six-seven split.

Frames from example animations appear in figure 3. The strike example is the easiest, because strikes are quite likely given our simulator. Bowling the six-seven spare is not difficult either, because the various solutions probably form a connected set in state space, so once a single solution is found, the others can be explored efficiently. Bowling the ball that leaves a six-seven split is the hardest example, intuitively because it is hard to knock down the pins behind the six pin while leaving it in place.

We also attempted to bowl the seven-ten split (figure 4). This shot depends on the precise frictional properties of the ball and lane. Our simulator's friction model could not capture the required effect (we
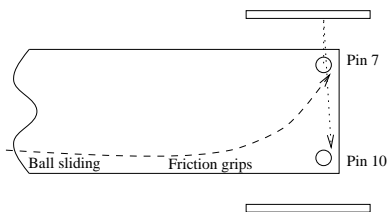

Figure 4: *The seven-ten split, in which the aim is to knock down both the seven and ten pins in one shot. The technique used by bowlers relies on the fact that a bowling ball will slide while spinning about an inclined axis, then, at some point, friction will cause the ball to grip, converting the angular momentum of the spin into linear momentum across the lane (dashed line). The seven pin must be struck behind its center of mass, so that it initially moves away from the ten pin (dotted line), bounces off the wall and moves back across the lane to hit the ten pin. Our simulator cannot model friction well enough to simulate this shot (we are not aware of any that can).*

are not aware of any that can), so we could not make the shot. This demonstrates that the MCMC algorithm will only generate samples that are plausible according to our model (section 4). Our simulation model says that balls never take really big hooks, so we never see animations involving big hooks, regardless of the constraints.

5.3 Balls that Spell

In these experiments we drop a stream of balls into a box partitioned into bins so that, when everything has come to rest, the balls form letters or symbols (figure 5). Details of the model appear in appendix B.1. We don't care which ball ends up in which designated bin. We use an impulse-based rigid-body simulator, as in the bowling example.

The uncertainty in this world arises from the shape of the partitions and the location from which each ball is dropped. The top surface of the partitions depends on a set of *partition vertices*, each of which is randomly perturbed about a default position. Each ball is dropped from a random location.

The constraint we impose is that, when all the balls have come to rest, each ball is in a designated bin. We fix the maximum number of balls, so if each ball falls into a designated bin there can be no ball in an undesignated bin. We face a situation in which we cannot propose animations that are certain to completely satisfy the constraints, so, as for the bowling example, we use the Gibbs distribution for the constraint probability $p_c(A) \propto \lambda^k$, where k is the number of balls in designated bins at the end of the animation.

To facilitate mixing we allow the number of balls in the simulation to vary between zero and the minimum number required to form the word, by flipping each ball between active and inactive states: inactive balls do not take part in the simulation. If all the designated bins are filled, removing a ball frees up a bin for another ball to move into, making a significant change to the animation. Removing the ball entirely, rather than just having it go into an undesignated bin, reduces the amount of interaction between the balls, possibly making it easier to make acceptable proposals. It also speeds the simulation when balls that aren't contributing anything are removed. Our initial experiments used a fixed number of balls, and the chain failed to mix well.

The probability of an animation depends on how many balls are participating, the initial locations of the balls and the offsets of each partition vertex (see appendix B.1).

5.3.1 Proposals

The proposal algorithm we use performs one of five actions:

Example 1 $\longrightarrow$
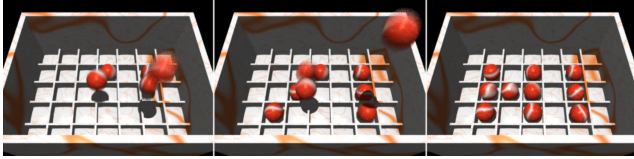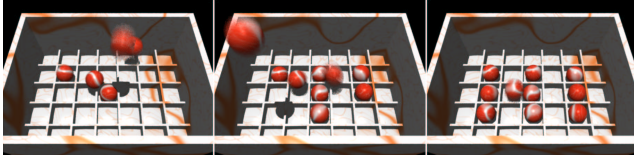

Example 2 $\longrightarrow$

Figure 5: *Two examples of the spelling balls model, in this case spelling "HI" in a seven by five grid. The shape of the boxes is allowed to vary slightly, as are the initial conditions of each ball. Our algorithm chooses box shapes and ball initial conditions that lead to the formation of a specific word.*

- The *change-all* strategy: change all the partition vertices and change all the balls.
- Change a subset of partition vertices.
- Change an active ball.
- Activate some balls (possibly none).
- Deactivate some balls (possibly none).

The change-all strategy appears as a means of escaping from low probability regions (figure 7). When an animation is found that satisfies the constraints, subsequent animations tend to also satisfy the constraints, but their probabilities degrade. This occurs because the reduction in probability for a partition vertex change may be quite small, and such proposals are likely to be accepted. The downward trend can continue, moving the chain into a region of low probability. Then, a change-all proposal can reset all the partition vertices to much higher probability values, and even though the constraints are no longer satisfied, the net change in probability will be positive and the proposal will be accepted. This *change-all effect* is good for mixing, because the next fully correct sample will generally be very different from the last.

The second and third proposals are designed to move around the state space by modifying balls or partitions, similar to proposals in previous examples. The proposals to activate or deactivate some balls let us change the number of balls in the simulation. The proposal strategy we use makes the probability of adding or deleting any given ball independent of the maximum number of balls. We first tried a proposal that chose a single ball and flipped its status, but if the maximum number of balls in the scenario is large, the probability of removing a ball goes up as more balls are activated while the probability of adding a ball goes down, making it difficult to get all the balls into the simulation.

The considerations in choosing a value for λ in this example are identical to those in the bowling example (a balance between good animations and good mixing), with an additional requirement due to the change-all effect: the constraint probability should be balanced against the model probability (in this case the probabilities of the partition vertices). If the constraint probability is too high, almost no change in partition vertices can overcome a well satisfied constraint. Good balance is achieved when a much better set of model values can overcome a constraint that is satisfied but uses poor model parameters.

As a specific example, we chose a bin designation that spells "HI" on a seven by five grid (figure 5). We used $\lambda = e^5$ for this word. A plot of k, the number of designated bins that are filled, for each
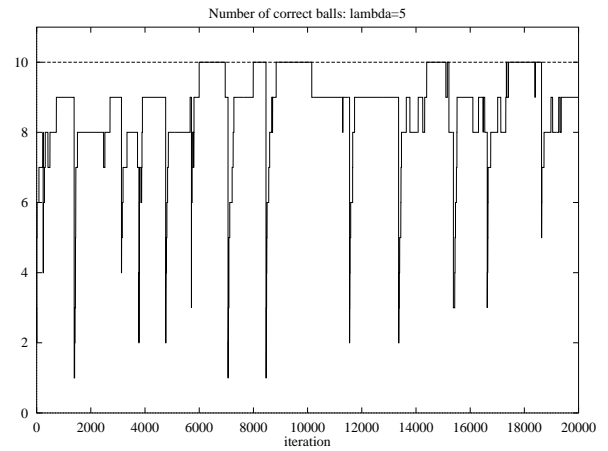

Figure 6: *The number of correctly positioned balls for each of twenty thousand iterations of the "HI" model, with $\lambda = e^5$. The maximum number of correct balls is ten. The chain finds its first good animation after around six thousand iterations (we have seen chains that find good animations within one thousand samples). This graph indicates good mixing because the chain spends only a short period of time near similar solutions, then makes significant changes before rapidly moving to a new good solution.*
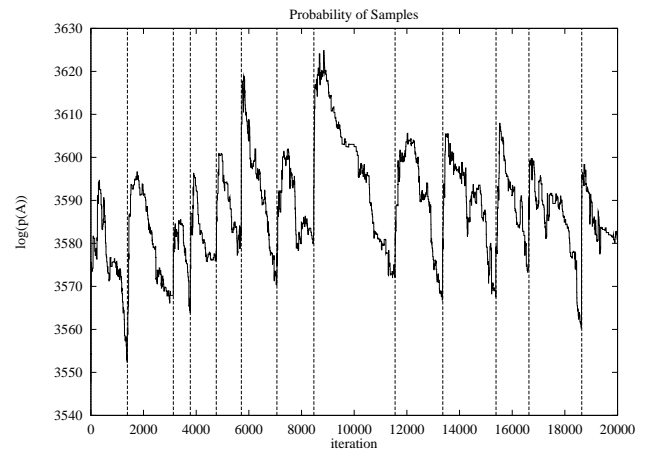

Figure 7: *The value of $\log(p(A))$ at each iteration of the chain in figure 6. The graph is quite bumpy, indicating good mixing. The dashed vertical lines correspond to all the iterations where the number of correct balls drops sharply (figure 6), yet all those iterations show a sharp rise in probability. This effect, due to the change-all proposal strategy, is discussed in the text.*

iteration of an example chain is shown in figure 6. The important feature of this graph is that the chain tends to rapidly reach correct spellings, stays there for a short period, then drops back to incomplete spellings. The twenty thousand iterations shown here took a few hours to compute on a 200MHz Pentium Pro PC.

The change-all effect is evident in this chain. Figure 7 plots the probability of the sample for each iteration. Places are marked where there is a sharp reduction in the number of correct balls, and these correspond to sharp increases in probability. At each of these sharp changes, a change-all proposal has been accepted that replaces a poor set of partition vertex offsets with a much more likely set, even though this breaks the constraint.

We experimented with different values of λ, both higher and lower, but they lead to less satisfactory chains. Values of λ that are

too low result in chains that have trouble finding correct animations, because the chance of accepting a poor proposal (from the point of view of the constraints) is too high. Values of λ that are too high make it less likely that a change-all proposal will be accepted, and also make it hard for the chain to abandon poor near-solutions. It takes only a few thousand iterations to see enough of the chain to know how lambda should be changed, and the range of acceptable values is reasonably large (our experiments show that chains with $\lambda = e^{5\pm1}$ are not much worse than those for $\lambda = e^5$) so little time must be spent in tuning parameters.

We also performed a larger experiment, with 30 of the 105 bins on a fifteen by seven grid to be filled (figure 8). In this example we used a value of $\lambda = e^{7.25}$ after experimenting with other values of λ between six and eight. The higher value for λ is required because there are more partition vertices and more balls. The greater number of partition vertices allow the change-all proposal to remain effective at higher λ values, so we still see adequate mixing. In fact, higher λ values are required to make it harder for a change-all proposal to succeed, so that the chain has enough time between major changes to converge to good animations.

5.4 Random Tables with Dice

This summarized example demonstrates objects bouncing on a random table, coming to rest in constrained configurations. Dice are used as random number generators in the real world because they are exceptionally hard to control [3], yet our technique is capable of finding animations in which dice come to rest near a particular place with a particular face showing.

The 2D ball example (section 5.1) used a very simple table model, with two main drawbacks due to the use of independent normals at each collision:

- An object bouncing in place will appear to have the table change underneath it as a different normal vector is chosen for each collision.
- Nearby points on the table are not correlated, as points on a real, bumpy table would be, which reduces the plausibility of the animations.

In this example we use a continuous, bumpy surface for the table. Rather than describe random normals directly, we specify a random b-spline surface via control points on a grid with fixed spacing but random vertical offsets. We can also specify random restitution and friction values at the control points, to be interpolated by the spline, thus extending the model to include the concept of springy or sticky regions on the table (such as spilt beer). The b-splines defining the table shape and properties define random fields over the surface. In principle, we could measure real tables, model their particular random fields, and use those in our simulation.

The simulator used in this example simulates only one object at a time bouncing on the random b-spline surface. It uses special techniques to manage the large number of control points required for a table with fine bumps.

In this example, constraints can be defined for any aspect of the object's 3D state at any point in time. Initial conditions for the object are specified by constraining its state at the start of the simulation ($t = 0$). The probability of an animation in this world contains components for the control vertices defining the table's shape, friction and restitution, and a component for each constraint on the object.

An animation generated from this type of scenario is shown in figure 9. Each of six dice is dropped and told to land in a specific place showing a specific side up. The dice are treated individually and do not interact — the table is not the same for each die. It took an hour or so of processing time to find a good animation for each die (a few hours for the complete animation). However, the chain does not mix well, so it takes many hours to find significantly different animations.

Figure 9: *A composite of six sample animations showing the control of a single bouncing die. Each die in the image was animated separately. Each had a different target location and desired side-up, but started with the same distribution on initial conditions.*

Proposals were made by changing one control point at a time, or one initial condition component at a time, or everything at once, the choice being made according to user supplied relative probabilities. Changes were made by adding a random offset to the current value, resulting in symmetric transition probabilities.

The ability to make changes at any point in the simulation, through the surface control points, makes it easier to find good animations in this world. Control points near the first few collisions get the die somewhere close to the target, and later collisions refine the location. This is not an explicitly coded strategy, rather it emerges naturally from the chain. However, a better proposal strategy might make explicit use of the behavior.

6 FUTURE WORK

The models we use arise naturally in the real world, and we provide a means of verifying the plausibility of simulations. With further work it should be possible to experimentally obtain more accurate models, and test simulation algorithms on such models, to obtain results like those of Mirtich et. al. [22].

It is an open problem to determine the difficulty of a particular example without experimentation. Computation time can be adversely affected because the simulation itself is slower, or more iterations are required to find good animations, or both. For example, our bowling and spelling ball examples take comparable times to compute, the former due to slow simulation and the latter due to difficult constraints. Simulation time dominates the cost of each iteration, so it is reasonable to spend more time making better proposals to improve mixing and hence reduce the total number of iterations. For example, in the bowling simulation we might bias changes in the ball's initial conditions according to which pins were knocked down.

Constraints in our approach are specified as probability density functions, which allows almost any type of constraint. In particular, it might be possible to constrain collisions or other events to occur at specific times (or frames). This would allow physically-based animations to be choreographed to music, or collisions to occur at frame boundaries.

Popović et. al. [24] describe an interactive algorithm for manipulating colliding bodies. As they suggest, a system might be designed to take as input animations generated by our MCMC approach and allow users to fine tune the outcome as desired using local, interactive operations.
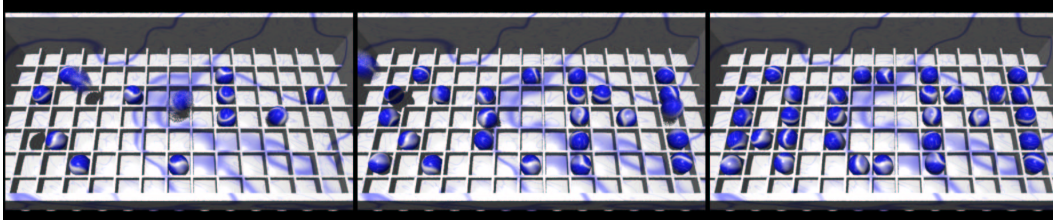
Example 1 ⟶
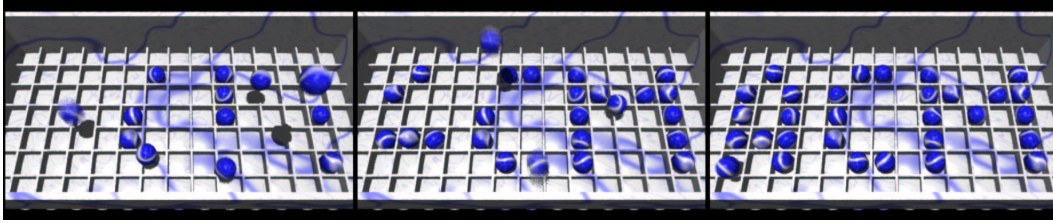
Example 2 ⟶

Figure 8: *Balls that spell ACM. The box contains 105 bins, of which 30 are designated to contain balls. We show two animations, one on each row, generated from a single chain. Each has the bins being filled in a different order, evidence that the chain produces a good mix of samples.*

We have only touched on the possibilities of plausible motion with constraints, focusing entirely on rigid body dynamics. Our techniques may also work in other domains that are hard to constrain, including group behaviors [4] and deformable objects [8]. Another goal is to develop real time systems in which specific events are forced to occur in a plausible manner. For example, in a computer game we might like the monster to surprise the player in a particular way, with a plausibility model that takes into account the viewer's knowledge of the monster's state and how it moves [7].

Acknowledgements

References

[1] Joel Auslander, Alex Fukunaga, Hadi Partovi, Jon Christensen, Lloyd Hsu, Peter Reiss, Andrew Shuman, Joe Marks, and J. Thomas Ngo. Further Experience with Controller-Based Automatic Motion Synthesis for Articulated Figures. *ACM Transactions on Graphics*, 14(4):311–336, October 1995.

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

[3] Ronan Barzel, John F. Hughes, and Daniel N. Wood. Plausible Motion Simulation for Computer Graphics Animation. In *Computer Animation and Simulation '96*, pages 184–197, 1996. Proceedings of the Eurographics Workshop in Poitiers, France, August 31-September 1, 1996.

[4] David Brogan and Jessica Hodgins. Group Behaviors for Systems with Significant Dynamics. In *Proceedings of the 1995 IEEE/RSJ International Conference on Intelligent Robots and Systems*, volume 3, pages 528–534, 1995.

[5] Lynne Shapiro Brotman and Arun N. Netravali. Motion Interpolation by Optimal Control. In *Computer Graphics (SIGGRAPH 88 Conf. Proc.)*, volume 22, pages 309–315, August 1988.

[6] Stephen Chenney. Asynchronous, Adaptive, Rigid-Body Simulation. SIGGRAPH 99 Technical Sketch. In Conference Abstracts and Applications, page 233, August 1999.

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

[8] Jon Christensen, Joe Marks, and J. Thomas Ngo. Automatic Motion Synthesis for 3D Mass-Spring Models. *The Visual Computer*, 13(3):20–28, January 1997.

[9] Michael F. Cohen. Interactive Spacetime Control for Animation. In *Computer Graphics (SIGGRAPH 92 Conf. Proc.)*, volume 26, pages 293–302, July 1992.

[10] Afonso G. Ferreira and Janez Žerovnik. Bounding the Probability of Success of Stochastic Methods for Global Optimization. *Computers and Mathematics with Applications*, 25(10):1–8, 1993.

[11] George S. Fishman. *Monte Carlo : concepts, algorithms, and applications*. Springer-Verlag, 1996.

[12] Walter R Gilks, Sylvia Richardson, and David J Spiegelhalter. *Markov Chain Monte Carlo in Practice*. Chapman & Hall, 1996.

[13] Michael Gleicher. Motion Editing with Spacetime Constraints. In *Proceedings 1997 Symposium on Interactive 3D Graphics*, pages 139–148, April 1997. Providence, RI, April 27-30.

[14] Radek Grzeszczuk and Demetri Terzopoulos. Automated Learning of Muscle-Actuated Locomotion Through Control Abstraction. In *SIGGRAPH 95 Conference Proceedings*, pages 63–70. ACM SIGGRAPH, August 1995.

[15] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. NeuroAnimator: Fast Neural Network Emulation and Control of Physics-Based Models. In *SIGGRAPH 98 Conference Proceedings*, pages 9–20. ACM SIGGRAPH, July 1998.

[16] Jessica Hodgins and Nancy Pollard. Adapting Simulated Behaviors for New Creatures. In *SIGGRAPH 97 Conference Proceedings*, pages 153–162. ACM SIGGRAPH, August 1997.

[17] Jessica K. Hodgins, James F. O'Brien, and Jack Tumblin. Perception of Human Motion With Different Geometric Models. *IEEE Transactions on Visualization and Computer Graphics*, 4(4):307–316, 1998.

[18] Mark Jerrum and Alistair Sinclair. Approximating the Permanent. *SIAM Journal of Computing*, 18:1149–1178, 1989.

[19] Mark Jerrum and Alistair Sinclair. The Markov Chain Monte Carlo Method: an approach to approximate counting and integration. In D.S.Hochbaum, editor, *Approximation Algorithms for NP-hard Problems*. PWS Publishing, Boston, 1996.

[20] Zicheng Liu, Steven J. Gortler, and Michael F. Cohen. Hierarchical Spacetime Control. In *SIGGRAPH 94 Conference Proceedings*, pages 35–42. ACM SIGGRAPH, July 1994.

[21] J. Marks, B. Andalman, P.A. Beardsley, W. Freeman, S. Gibson, J. 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 *SIGGRAPH 97 Conference Proceedings*, pages 389–400. ACM SIGGRAPH, August 1997.

[22] Brian Mirtich, Yan Zhuang, Ken Goldberg, John Craig, Rob Zanutta, Brian Carlisle, and John Canny. Estimating Pose Statistics for Robotic Part Feeders. In *Proceedings 1996 IEEE International Conference on Robotics and Automation*, volume 2, pages 1140–1146, 1996.

[23] J. Thomas Ngo and Joe Marks. Spacetime Constraints Revisited. In *SIGGRAPH 93 Conference Proceedings*, pages 343–350. ACM SIGGRAPH, August 1993.

[24] Jovan Popović, Steven Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive Manipulation of Rigid Body Simulations. In *SIGGRAPH 2000 Conference Proceedings*. ACM SIGGRAPH, July 2000.

[25] Zoran Popović and Andrew Witkin. Physically Based Motion Transformation. In *SIGGRAPH 99 Conference Proceedings*, pages 11–20. ACM SIGGRAPH, August 1999.

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

[27] Alistair Sinclair, 1999. Personal communication.

[28] Richard Szeliski and Demetri Terzopoulos. From Splines to Fractals. In *Computer Graphics (SIGGRAPH 89 Conf. Proc.)*, volume 23, pages 51–60, July 1989.

[29] Diane Tang, J. Thomas Ngo, and Joe Marks. N-Body Spacetime Constraints. *The Journal of Visualization and Computer Animation*, 6:143–154, 1995.

[30] Eric Veach and Leonidas J. Guibas. Metropolis Light Transport. In *SIGGRAPH 97 Conference Proceedings*, pages 65–76. ACM SIGGRAPH, August 1997.

[31] Baba C Vemuri, Chhandomay Mandal, and Shang-Hong Lai. A Fast Gibbs Sampler for Synthesizing Constrained Fractals. *IEEE Transactions on Visualization and Computer Graphics*, 3(4):337–351, 1997.

[32] Andrew Witkin and Michael Kass. Spacetime Constraints. In *Computer Graphics (SIGGRAPH 88 Conf. Proc.)*, volume 22, pages 159–168, August 1988.

A Bowling Details

A.1 Uncertainty model

Our bowling model is derived from data found online (http://www.brittanica.com). The sources of uncertainty in the model are:

Ball radius Distributed uniformly on $[r_{min}, r_{max})$, where r_{max} is specified by the rules of the game (approximately 11cm) and $r_{min} = 0.8 r_{max}$.

Ball density Distributed uniformly on $[\rho_{min}, \rho_{max})$, with $\rho_{max} = 1440 \text{ kg m}^{-3}$ and $\rho_{min} = 0.8 \rho_{max}$.

Ball initial position Fixed in line with the end of the lane, at some point uniformly distributed across the width of the lane, and at a height distributed according to $\Phi(1.1 r_{max}, 0.1 r_{max})$, the distribution defined by the Gaussian density function with mean $1.1 r_{max}$ and standard deviation $0.1 r_{max}$.

Ball initial velocity The component down the lane (measured in m s^{-1}) is distributed according to $\Phi(7.0, 1.0)$. The component across the lane is distributed according to $\Phi(0.0, 0.1)$. The vertical component is set to zero.

Ball initial angular velocity About a vertical axis (measured in rad s^{-1}), distributed according to $\Phi(0.0, 0.2)$.

Each pin Fixed shape and mass, offset from its proper location on the lane in a random direction by a distance distributed according to $\Phi(0.0, 1.0)$ (mm).

With these distributions:

$$
p_w(A) \quad \propto \quad e^{-\frac{1}{2}\left(\frac{x_z - 1.1 r_{max}}{0.1 r_{max}}\right)^2} e^{-\frac{1}{2}(v_x - 7)^2} e^{-\frac{1}{2}\left(\frac{v_y}{0.1}\right)^2} \times
$$
$$
e^{-\frac{1}{2}\left(\frac{\omega_z}{0.2}\right)^2} \sum_{pins} e^{-\frac{1}{2}\left(\frac{d_i}{0.001}\right)^2}
$$

where x_z is the ball's height above the lane, v_x and v_y are the velocity components down and across the lane, ω_z is the angular velocity about the vertical axis, and d_i is the distance of pin i from its center location. The above formula for $p_w(A)$ is valid if all the uniformly distributed variables are within their range, and all the fixed variables have their correct values, otherwise $p_w(A) = 0$. We can ignore the uniformly distributed variables in computing $p_w(A)$ because their distribution function is proportional to one.

A.2 Proposals

Our proposal mechanism samples a value u, uniform on $[0, 1)$, and then:

- if $u < 0.05$, we sample new values for all the random variables.

- if $0.05 \leq u < 0.125$, we change the radius of the ball by adding an offset distributed according to $\Phi(0.0, 0.04 r_{max})$. If the radius lies outside the allowable range, we wrap it back into the range.

- if $0.125 \leq u < 0.2$, we change the density of the ball by adding an offset distributed according to $\Phi(0.0, 0.04 \rho_{max})$, wrapping to keep ρ inside the allowable range.

- if $0.2 \leq u < 0.4$, we change the initial position of the ball. We add a horizontal offset distributed according to $\Phi(0.0, 0.2 w)$ (w the width of the lane), wrapping if necessary, and add a vertical offset distributed according to $\Phi(0.0, 0.05 r_{max})$.

- if $0.4 \leq u < 0.6$, we change the initial velocity of the ball. To its component down the lane, we add an offset distributed according to $\Phi(0.0, 0.5)$. To its component across the lane, we add an offset distributed according to $\Phi(0.0, 0.05)$.

- if $0.6 \leq u < 0.8$, we change the initial angular velocity by adding an offset distributed according to $\Phi(0.0, 0.1)$.

- otherwise, for each pin, with probability $\frac{1}{2}$, change its location by moving it in a random direction by a distance distributed according to $\Phi(0.0, 0.5)$ (mm).

All of these proposals are symmetric, so there is no need to compute the transition probabilities (they cancel when computing the acceptance probability).
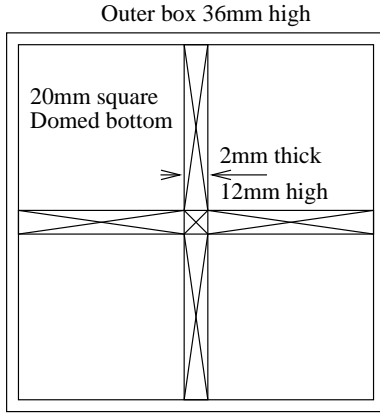
Outer box 36mm high

20mm square
Domed bottom

2mm thick
12mm high

Figure 10: *The dimensions and tesselation for the box in the spelling ball example.*

B Spelling Ball Details

B.1 Model details

Each bin has a side length of 20mm, and each partition is 2mm thick and 12mm high (figure 10). The floor beneath each bin is domed to help the balls come to rest, and the box in which the bins sit is 36mm deep. Each partition vertex is offset in a random direction by a distance distributed according to $\Phi(0.0, 0.1)$ (mm). Each ball is dropped from rest at a uniformly random location within a rectangle 72mm above the bottom of the box and centered above it. The size and density of the balls is intended to resemble marbles.

The probability of an animation is:

$$p(A) \propto \lambda^k \prod_{\mathcal{V}} e^{-\frac{1}{2}\left(\frac{\|\mathbf{x}_v\|}{0.1\,\mathrm{mm}}\right)^2} \prod_{\mathcal{B}} \frac{1}{a}$$

where k is the number of designated bins that are filled, $\mathcal{V}$ is the set of partition vertices, $\mathbf{x}_v$ is the offset distance of vertex v (in mm), a is the area of the rectangular region from which the balls may be dropped and $\mathcal{B}$ is the set of active balls, which can vary in size for different animations as balls are activated or deactivated. In this case we must include a term for the uniformly distributed drop position of each ball because the number of balls can vary.

B.2 Proposals

Our proposal mechanism uniformly samples a random $u \in (0, 1]$, and then applies one of four strategies:

- if $u < 0.04$, we change all the partition vertices, giving them new randomly chosen offsets, and change all the balls, giving them a new active status and a new initial position.

- if $0.04 \leq u < 0.36$, for each partition vertex, we randomly decide, with probability 0.02, to change its location by adding an offset in a random direction with length distributed according to $\Phi(0.0, 0.05)$ (mm).

- if $0.36 \leq u < 0.68$ we uniformly randomly select an active ball to change, and offset its starting position in a random direction for a distance distributed according to $\Phi(0.0, s)$, where s is the bin size. We wrap the edges of the region from which balls may be dropped.

- if $0.68 \leq u < 0.84$, for each enabled ball we uniformly sample $v \in (0, 1]$ and disable the ball if $v < 0.25$.

- otherwise, for each disabled ball we uniformly sample $v \in (0, 1]$ and enable the ball if $v < 0.25$.

All but the last two proposals are symmetric. If a ball is disabled, the ratio $\frac{q(A_i \mid A_c)}{q(A_c \mid A_i)} = a$ (the area of the rectangle from which the ball may be dropped). If a ball is enabled, $\frac{q(A_i \mid A_c)}{q(A_c \mid A_i)} = \frac{1}{a}$.