STATISTICAL AND DIRECTABLE METHODS FOR LARGE-SCALE RIGID BODY SIMULATION

A Dissertation

by

SHU-WEI HSU

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

May 2013

Major Subject: Computer Science

ABSTRACT

This dissertation describes several techniques to improve performance and controllability of large-scale rigid body simulations. We first describe a statistical simulation method that replaces certain stages of rigid body simulation with a statistically-based approximation. We begin by collecting statistical data regarding changes in linear and angular momentum for collisions of a given object. From the data, we extract a statistical "signature" for the object, giving a compact representation of the object's response to collision events. During object simulation, both the collision detection and the collision response calculations are replaced by simpler calculations based on the statistical signature. In addition, based on our statistical simulator, we develop a mixed rigid body simulator that combines an impulse-based with a statistically-based collision response method. This allows us to maintain high accuracy in important parts of the scene while achieving greater efficiency by simplifying less important parts of the simulation. The resulting system gives speedups of more than an order of magnitude on several large rigid body simulations while maintaining high accuracy in key places and capturing overall statistical behavior in other places.

Also, we introduce two methods for directing pile behavior to form the desired shapes. To fill up the space inside the desired shapes and maintain the stability of the desired pile shapes, our methods analyze the configurations and status of all objects and properly select some candidates to have their degrees of freedom (DOFs) reduced. Our first method utilizes the idea of angles of repose to perform the analysis. According to the desired angle of repose, we create an additional spatial structure to track the piling status and select suitable objects to reduce their DOFs. In our second method, we adapt equilibrium analysis in a local scheme to find “stable” objects of the stacking structure. Then, we restrict their DOFs by adding constraints on them for stabilizing the structure. Overall, our directing methods generate a wider variety of piled structures than possible with strict physically-based simulation.

ACKNOWLEDGEMENTS

First and foremost, I want to thank my advisor, John Keyser, for his advice, encouragement, and support throughout my time as his student. In particular, I am grateful that I have the chance to take his 441 course since his teaching sparked my interest in computer graphics.

I would like to thank the other members of my committee. All of them provides ideas, directions, and improvements for my dissertation.

I particularly want to thank Dunghui Han, who conducted the major part of the user study in this dissertation. I also want to thank the other students that I have the opportunity to work with. Daniel Miller, Ruoguan Huang, Songgang Xu, Billy Clack, Josiah Manson, Lei He, and Jason Smith, I have had a great time in the office with all of you.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

LIST OF TABLES

TABLE Page

1. INTRODUCTION

In our world, people are very often involved in scenarios having large numbers of objects. Figure 1.1 shows a couple of examples: collecting thousands of products from a farm, playing with plastic balls in a ball pit, or recycling a large number of bottles. In these examples, many complex dynamic behaviors are occuring. Individually, an object could collide, roll, slide, and rebound into another one; entirely, all objects might be stacked together to fill spaces or to form piles. The more objects a scenario has, the more complex it will be.

In computer graphics, simulation technologies provide an opportunity to bring these scenarios to the digital world. It is so successful that we have seen a lot of similar scenarios in video games, cartoon animations, digital art, and feature films. The success also brings more demands. Undoubtedly, enlarging the scale of the simulation is one of the more desired demands. For example, visual effects artists may want to enlarge the scale of a scenario to an incredible level, making the digital content more magnificent and exciting. Game developers often want to simulate more objects in games and keep the realtime performance as well. These demands post a challenge, performance. If performance of the simulation methods is not fast enough, the scale will be very limited. Another demand is the ability to direct simulations. To create attractive content, a visual effects artist may want to direct simulations. They need to manipulate the simulation to support the needs of the art styles or the needs of the stories. For example, they may need to guide the moving path of a simulated object, or guide a group of objects to pile up with a desired shape. However, the techniques for directing small scale simulations may not be practical for directing large scale simulations. From these observations, we see challenges remain for efficiently handling and controlling large scale multibody simulations. Certainly,

Figure 1.1: Many scenarios in our society involve large numbers of objects. Top: A whole truckload of lemons. Mid: Large numbers of balls in a playing pit. Bottom: Numerous amounts of recycled bottles. All image are licensed under Creative Commons and are taken by the following Flickr users: Susie Wyshak(Top), Mayhem (Middle), and Michal Osmenda (Bottom).

the objects in multibody simulation could be granular materials such as corn kernels, rigid objects such as bricks, slightly deformable objects such as tires, or very soft objects such as clothes. In this dissertation, however, we focus our attention on rigid objects.

In this dissertation, we describe new technologies for advancing performance and controllability of large-scale rigid body simulations. First, we develop a statistical rigid body simulation system, which efficiently handles large-scale rigid body simulation via statistical methods. Second, we introduce two simulation control methods for directing piling behaviors, one of the more interesting, and challenging, features of large rigid body simulations.

1.1 Rigid Body Simulation

For more than a decade, the need for rigid body simulations has increased in different domains, such as engineering and entertainment. Also, rigid body simulation occurs across many different computational devices, from supercomputers to personal mobile devices. The need for rigid body simulation also drives much research for improving simulation technologies.

Today, although we have better computers and better simulation technologies, the need for fast simulation approaches keeps going due to the demands of simulation scales that grow larger and larger. Many current simulation techniques are too slow to be used for real-time applications, or are even too slow to be used for many offline applications. In addition, the rise of small portable devices is another reason to keep looking for faster simulation methods. The computational capabilities of those devices are very limited, but the demands of running rigid body simulations for games that run on them are very high.

Improving performance for large-scale rigid body simulations is challenging. Generally, there are three important steps in rigid body simulation: motion status integration, collision detection, and collision response. Since the complexity of collision detection algorithms are not linear, performance slows down very quickly when the number of simulated objects increases. Also, when stacking happens, performance gets worse since a huge number of contacts could keep occurring every time step. Much research has shown that simplified models (of geometry, contact, or friction) and GPU-based methods are able to increase performance. In these methods, however, collision detection is still the bottleneck, blocking the path for reaching larger scale simulations. So, handling collisions efficiently is key to conquering performance challenges in large-scale simulations.

In certain situations, people will not be able to predict exactly correct results or notice some minor errors of natural phenomena and physical effects. For example, seeing a rigid object dropped to a desk, one might expect the object will bounce and roll, but have little sense of the exact response at each bounce. In addition, an enormous number of events happen in parallel in a complex simulation scenario. It is possible that we can simulate the plausible look and feel of complex situations without making every single event totally correct.

On the basis of these insights, many simplified simulation techniques and concepts such as simulation levels of detail have been developed. However, most of these techniques have tended to use the same simulation framework, reduce the problem, and then compute it with less fidelity (e.g. using a coarse level of geometric detail).

With the huge success of statistics in many computer science fields, such as search engines and social networks, we think it is possible to increase performance of rigid body simulation via statistically-based approaches. We want to discover how to use statistical techniques to improve the computational performance of large-scale simulations.

1.2 Controllability for Rigid Body Simulation

Not just performance but also controllability is very important to rigid body simulations. Results from simulations may not always present a desired result. Users are eager to guide the simulation process to create more attractive digital content; for example for an object to land with a specific orientation or follow a designed bouncing path. Simply providing some tunable parameters in the dynamics equations cannot achieve this task. Therefore, many studies have introduced novel control methods, allowing artists to easily manipulate the simulation processes to achieve the results they want to present. One example is keyframe-based approaches. A user could define some key frames for an object, and the controlling algorithm and the simulation algorithm will work together to force the simulated path to interpolate through those key frames, while keeping the result plausible.

However, it is still very challenging to provide controllability for large scale simulation. Most of the controlling methods still target small scale problems due to performance. Many controlling algorithms have been proposed based on optimization approaches, which are extremely slow and cannot be used in a system with a large number of objects. In addition, large-scale simulations might need different types of controls; for example, having thousands of object stack up and form an exaggerated shape. Developing intuitive and practical controls for users to manipulate large-scale simulations is still an interesting open topic.

We noticed that there is a specific phenomenon, piling behavior, which demands tweaking in large-scale rigid body simulations. Recently, we have seen a lot of piles of objects in many animation films: for instance, trash piles in *Wall-E*, food piles in

Ratatouille and *Cloudy with a Chance of Meatballs*, and stone piles in *Toy Story 3*. In those shots, large numbers of objects are stacked together to compose piles. The shapes of those piles vary a lot, from flat to tall, and from realistic to exaggerated. It is not easy to freely tweak the shapes of the piles by only relying on the friction and the contact modeling techniques. This is simply because there is no direct mapping between those dynamics models and the shape styles.

There exist some methods that can create aggregates with different styles statically, such as procedural based approaches[59] or synthesis based approaches[46]. In addition, in a course section of SIGGRAPH '07, Cho[13] shared a story about how they created a food pile in *Ratatouille*. Still, we haven't seen a method that can efficiently direct the piling behavior in a dynamic simulation environment. This motivated us to design algorithms that can direct the piling process to form the shape that a user want.

1.3 Thesis Statement

My thesis statement is:

Simulating large numbers of rigid bodies can be made efficient by modeling the problem with a statistical computation scheme, and be made art-directable for stacking behavior through degree-of-freedom reduction techniques.

One statistically-based rigid body simulator and two stacking control algorithms are designed, implemented, and examined to support this thesis statement.

The key ideas behind the statistical computation scheme are precomputation and reuse. A large amount of simulation data will be captured, analyzed, and arranged to build statistical models of collision response behaviors in advance. Then, at runtime, we avoid the necessity of accurate collision determination by replacing impulse-style collision response with a statistically based response model.

To direct the shape during the pile processing, our key idea is to properly select objects and reduce their degrees of freedom (DOFs). The purpose is to make the desired space be occupied and to stabilize the pile structure. To make sure the animation results are plausible, we have to carefully pick objects to reduce their DOFs. For example, if an object is already in the interior of a pile, freezing it won't affect the plausibility of the simulation. Therefore, we have to evaluate whether a DOF reduction action is safe, with respect to the visual quality, before we do that. To do so, we consider an object's physical properties and several characteristics of piling behavior, such as angles of repose, space distributions, and equilibrium conditions of the structures, to select candidates. Then, the control process won't break plausibility of simulation results.

1.4 Research Overview

In this dissertation, we describe new methods to solve the efficiency and controllability problems when simulating large numbers of rigid bodies. The new methods include one statistically-based rigid body simulator and two stacking control algorithms. Much of this work has appeared in prior publications[28][29][30].

Our statistically-based rigid body simulator is a fast rigid body simulator that uses a novel statistical approach to efficiently simulate large numbers of rigid objects. Rather than performing a full simulation, this method instead replaces key aspects of the simulation with results generated to mimic the statistical behavior of the simulation. The method has two stages. In the first stage, which is precomputed, we extract an object's response to collisions from a lot of examples with a statistical representation, called statistical signatures. In the second stage, during run time, we use the statistical signature to quickly compute collisions and responses of that object type. Note that the method is appropriate for simulation situations requiring significant performance improvement, and allowing for some loss in fidelity. Evaluations show the new method boosts performance to a level that other simplified methods cannot achieve. Meanwhile, it generates plausible results.

Moreover, we broaden the usability of the statistical rigid body simulation method by combining it with a physically-based rigid body simulator. We can specify different simulation methods according to an object's visual importance or different necessity in the scene. The goal of this part is to offer a nonuniform simplification, creating plausible rigid body simulations and achieving better performance as well.

In the second part of the dissertation, we describe two algorithms for stacking control. First, we design a simple control method to support the art-directable pile modeling for conical structures. This method uses the idea of angle of repose to perform stacking control. Angles of repose determine the overall look-and-feel of conical pile structures. To model different styles of piles, a user can simply specify the angles of repose of the pile shapes. The control algorithm will automatically track and control the stacking behavior to adjust the shape during simulation. Our method allows users to specify the shapes of piles that can be obtained, allowing for a more realistic or more art-directed look to emerge.

In addition, we present a more intuitive simulation control algorithm to support art-directable stacking designs by automatically adding constraints to stabilize the stacking structure. Unlike in the first control method, the structure can be any shape, as long as it is physically achievable. We take users' final designs, which might have stability problems as input. Then, our method strengthens their designs by carefully decreasing the degrees of freedom (DOFs) of objects during simulation. This helps reduce the perturbation in the structure and helps preserve the volume of the shape as well.

Altogether, the new techniques presented in this dissertation allow us to achieve significant performance improvement, and provide more art-directable stacking behaviors in large-scale rigid body simulations.

1.5 Chapter Overview

The rest of this dissertation is organized as follows. In Chapter 2, we review several background and related studies. In Chapter 3, we describe the theory, implementation, and evaluations of our statistically-based rigid body simulator. In Chapter 4, we discuss how to broaden the usability of a statistically-based simulator by combining it with a physically-based simulation. In Chapters 5 and 6, we describe the two stacking control algorithms. Finally, in Chapter 7, we discuss the advantages and limitations of the methods we presented here, as well as the possible future extensions.

2. BACKGOUND

In this chapter, we provide background material which helps readers to understand the later chapters. We start by reviewing the fundamental of rigid body simulation. Next, we will talk about the problems of object aggregation. Finally, we will discuss several related studies.

2.1 Rigid Body Simulation

Rigid body modeling can be decomposed into four steps, motion state integration (predict the new motion state of an object in the next timestep), collision detection (finding whether two objects collide), collision determination (for two colliding objects, finding the point(s) where the collision is occurring), and collision response (applying some force such as an impulse to response to the collision).

2.1.1 Motion State Integration

To predict the motion state of a rigid body, we need to compute the changing rate of the motion state at current timestep. Here we roughly describe the procedures of the integration step. The derivation of the equations in this section can be found in [4] and [50].

First, each object has two constant variables, mass, m, and the inertia tensor of the local object space, I_{local}. Also, we need to assign an initial motion state to an object. To represent the motion state of a rigid body, we need four variables,

$$Y = \begin{pmatrix} \mathbf{x} \\ R \\ P \\ L \end{pmatrix}. \tag{2.1}$$

The position of the object is denoted as $\mathbf{x}$, a vector $\in \Re^3$. The orientation of the object is denoted as R, a 3×3 rotation matrix. P is the linear momentum and L is the angular momentum (P and L $\in \Re^3$).

Three derived states can be computed directly from the motion state:

$$\mathbf{v} = \frac{P}{m} \tag{2.2}$$

$$I^{-1} = R I_{local}^{-1} R^T \tag{2.3}$$

$$\mathbf{w} = I^{-1} L \tag{2.4}$$

$\mathbf{v}$ and $\mathbf{w}$ are the linear and angular velocity of the object, respectively. I^{-1} is the inertia tensor of the global space.

The derivative of Y (the changing rate of the motion state), $\dot{Y}$, can be computed by the following equations:

$$\dot{\mathbf{x}} = \mathbf{v} \tag{2.5}$$

$$\dot{R} = \mathbf{w}^* R \tag{2.6}$$

$$\dot{P} = F \tag{2.7}$$

$$\dot{L} = T \tag{2.8}$$

Here $\mathbf{w}^*$ is a matrix made of the components of $\mathbf{w}$. This is a mathematical techniques for reforming a cross product to dot product. Basically, $\mathbf{w}\times R$ is equal to $\mathbf{w}^*R$.

$$\mathbf{w}^* = \begin{pmatrix} 0 & -w_z & w_y \\ w_z & 0 & -w_x \\ -w_y & w_x & 0 \end{pmatrix} \tag{2.9}$$

F and T are the total force and the total torque that work on the rigid body. The sources of the them could be gravity (only for the total force), contact forces, or users' manipulations.

In the integration procedure, we first accumulate the forces and torques for each object. Then we compute $\dot{Y}$, the changing rate of the motion state. Finally we use an ODE integrator (e.g. Euler method or Runge-Kutta method) to compute the new motion state in the next timestep, as well as these three derived states ($\mathbf{v}$, I^{-1}, and $\mathbf{w}$).

2.1.2 Collision Detection and Determination

Generally, collision detection has two phases, "broad phase" and "narrow phase". The goal of the broad phase is to quickly determine which pairs of objects "might" be intersecting. On the other hand, in the narrow phase, more precise tests will be applied to determine whether a pair of objects is intersecting.

For the broad phase collision detection, spatial subdivision and sweep-and-prune are popular methods. Spatial subdivision approaches divide the whole scene into small cells and then check overlapping between the objects and the small cells. If one cell is occupied by more than one object, there is a potential collision. Moreover, it is ideal to use hash tables to accelerate the tests. The sweep-and-prune approach first gives bounding volumes to all objects. A bounding volume is a simplified 3D shape such as a sphere, an axis-aligned bounding box (AABB), or an oriented bounding box (OBB), which is used to completely encapsulate the selected part of the object. Then, the approach projects the lower bounds and upper bounds of all objects to one axis (could be more than one). Next, by sorting and going through the list of the lower bounds and upper bounds on the axis, potential collisions could be found. The update of the list can be very efficient due to the timestep-to-timestep coherence. Because objects generally don't move too far from timestep to timestep, only a few elements on the bounding list will change their order. Based on this characteristic, insertion sort can be used to maintain the list very efficiently. The brute force approach, which tests all pairs, needs to conduct $n(n-1)/2$ collision tests for n objects. Obviously the complexity is $\mathrm{O}(n^2)$. With spatial subdivision approach or sweep and prune algorithm, the expected complexity is way much better than $\mathrm{O}(n^2)$ in temporally coherent environment [66], though the worst-case is still $\mathrm{O}(n^2)$.

For the narrow phase, bounding volume hierarchy (BVH) techniques are popular approaches for speeding up collision tests. A bounding volume is a simplified 3D shape such as a sphere, axis-aligned box, or oriented box, which is used to completely encapsulate the selected part of an object. There are two main purposes of using a bounding volume hierarchy. First, it provides cheaper cost for testing overlaps. Second, it allows users to find non-colliding situations faster. To create the hierarchy, top-down and bottom-up method are both popular strategies. For the top-down method, it first creates a larger bound volume to cover the whole object. Then, it divides the object into several parts, creates smaller bounding volumes to cover these parts, and organizes the the hierarchies of these smaller bounding volumes. Usually a tree structure is used to organize the hierarchies. The previous step will be repeated until each bounding volume only includes one primitive element (e.g. a triangle). With a hierarchical representation of these bounding volumes, a collision test can be terminated earlier when no collision is found at current hierarchy.

In addition, distance tracking algorithms are favorable for accelerating the tests of BVH pairs. These kinds of algorithms can also use timestep-to-timestep coherence to efficiently track the closest points on features (vertices, edges, or faces) of a pair of objects. These algorithms rely on two key ideas. First, use Voronoi regions to quickly identify the closest feature to a given point. The property of Voronoi partitions can help quickly determine the closest pair of features. Second, start the test from the closest feature pair in the previous timestep. Again, since objects usually don't move too much from timestep to timestep, the closest feature pair in the previous timestep is highly likely to be the closest pair of this timestep. Lin-Canny's algorithm[41] is one of the famous feature-based methods. In collision detection libraries such as I-COLLIDE[14] or SWIFT++[18], Lin-Canny's algorithm is used to tested BVHs pairs.

Several pieces of information that are needed for resolving the collision should be calculated after a collision found. For example, the contact position, contact normal, penetration depth, and so on. This step is usually called collision determination. These computations are naturally combined within the narrow phase collision detection.

2.1.3 Collision Response

Once a collision is found, we need to immediately apply some forces (such as impulses) on the objects to respond to the collision, avoiding the inter-penetration problem.

2.1.3.1 Separating collision contacts and resting contacts

Moore and Wilhelms[51] proposed a popular scheme to handle the contacts in rigid body simulation — splitting "colliding contacts" and "resting contacts". Because they have different characteristics, solving them separately is easier. So, under this scheme, a general a simulation loop runs like this: process colliding contacts, integrate motion states, and then handle resting contacts. The main difference between colliding contacts and resting contacts is the relative velocity of two objects at the contact point. When a contact happens, if the relative velocity at the contact point indicates that these two objects are approaching each other, it will be labeled as a colliding contact. Usually this happens when two objects hit into each other with certain speeds, and then they will rebound immediately. On the other hand, resting contact happens when the relative velocity at the contact point is close to zero. This happens when one object is just touching, but neither penetrating to nor separating from another object; for example, an object resting or sliding on another one.

2.1.3.2 Solve colliding contacts

The goal of handling colliding contacts is to instantaneously change the velocity of the colliding objects to prevent penetration. To compute new velocities for the objects, a general approach is to formulate the change of momentum (before contact and after contact), which is called impulse, by the restitution equation and the idea of conservation of momentum. After solving for the impulse, we apply it on both objects to change their velocities. We will discuss the computation of impulses in 2.1.4.

2.1.3.3 Solve resting contacts

The goal of handling resting contacts is similar to handling colliding contacts, but it has a slight difference. The resting behavior should be maintained; otherwise artifacts will degrade the visual quality; for example, an object might jitter on the ground. Roughly, three different types of approaches have been used to solve resting contacts, penalty method[51][17], analytical method[3], and impulse method[50][24]. The analytical method is a global method, which means it solves all resting contacts at a time. On the contrary, the penalty and impulse methods are local methods. They solve the contacts one by one.

The simplest penalty method applies spring forces on the contact points to minimize the penetrations. The computation cost is very low, and the implementation is very easy too. A common issue of the method is the stiffness of the spring. A stiff spring would make objects jitter, but a soft spring would let them penetrate into other objects.

The analytical method is a constraint-based method. By imposing a zero penetration distance as the constraint, it computes the correct contact forces at the contact point to avoid penetration. Usually it can be modeled as a linear complementarity problem (LCP). Non-penetration is guaranteed if we solve the problem correctly. However, when considering frictions in the system, it becomes very hard to solve (NP-Hard)[17]. Also, it is more difficult to implement.

The impulse method basically computes impulses for both colliding contact and resting contact. In other words, it treats all contacts the same. But, it needs special treatments to avoid penetrations or jittering effects when it handles resting contacts. For example, in Mirtich's thesis[50] he dynamically determined the coefficient of restitution to relieve the penetration problem. The more penetration an object got, the higher of a coefficient should be used. Furthermore, Guendelman et al. [24] proposed an alternative way to avoid the penetration. Their main idea is to postpone the position integration until all contacts are solved. After handling colliding contacts, they just integrate the velocity states. If the position is updated here, the updated velocity will mess up the resting behaviors. So, they immediately solve the resting contacts, which will correct the velocity. After this, it is safe to update the position states.

2.1.4 Impulse Computation

Here we review the impulse computation with static and kinetic friction.

To goal is to find an impulse **j** at a point, **c**, to change the velocity to avoid further penetration. Let **r** be the vector from **c** to the center of mass of the object. The change of linear and angular velocities can be described in the following equations,

$$\mathbf{v}^+ = \mathbf{v}^- + \frac{\mathbf{j}}{m} \tag{2.10}$$

$$\mathbf{w}^+ = \mathbf{w}^- + I^{-1}(\mathbf{r}^*\mathbf{j}) \tag{2.11}$$

The $+$ notation means new velocity; the $-$ means old velocity. Again, we convert $\mathbf{r} \times \mathbf{j}$ to $\mathbf{r}^*\mathbf{j}$.

So, the new and old combined velocities at the contact point, $\mathbf{u}^+$ and $\mathbf{u}^-$, can be described in the following equations,

$$\mathbf{u}^- = \mathbf{v}^- + \mathbf{w}^- \times \mathbf{r} \tag{2.12}$$

$$\mathbf{u}^+ = \mathbf{v}^+ + \mathbf{w}^+ \times \mathbf{r} \tag{2.13}$$

By plugging (2.10) and (2.11) in to (2.13), we can have

$$\begin{aligned}
\mathbf{u}^+ &= \mathbf{v}^- + \frac{\mathbf{j}}{m} + (\mathbf{w}^- + I^{-1}(\mathbf{r}^*\mathbf{j})) \times \mathbf{r} \\
&= \mathbf{v}^- + \frac{\mathbf{j}}{m} + \mathbf{w}^- \times \mathbf{r} + (I^{-1}(\mathbf{r}^*\mathbf{j})) \times \mathbf{r} \\
&= \mathbf{u}^- + \frac{j}{m} + \mathbf{r}^{*T}(I^{-1}(\mathbf{r}^*\mathbf{j})) \\
&= \mathbf{u}^- + (\frac{\mathbb{1}}{m} + \mathbf{r}^{*T}(I^{-1}(\mathbf{r}^*)))j \\
&= \mathbf{u}^- + K\mathbf{j} \qquad (2.14)
\end{aligned}$$

Next, we are going to list the new and old relative velocities of two objects at the contact point so that we use Newton's law of restitution to solve the impulse value. Also, we know the impulse works on both objects with opposite directions but with the same magnitude. The relative velocity can be described as the following equations,

$$\begin{aligned}
\mathbf{u}_{rel}^- &= \mathbf{u}_a^- - \mathbf{u}_b^- \qquad (2.15) \\
\mathbf{u}_{rel}^+ &= \mathbf{u}_a^+ - \mathbf{u}_b^+ \\
&= (\mathbf{u}_a^- + K_a\mathbf{j}) - (\mathbf{u}_b^- + K_b\mathbf{j}) \\
&= (\mathbf{u}_a^- - \mathbf{u}_b^-) + (K_a + K_b)\mathbf{j} \\
&= \mathbf{u}_{rel}^- + K_T\mathbf{j} \qquad (2.16)
\end{aligned}$$

Generally, restitution equation can be described as

$$-\epsilon \mathbf{u}_{rel}^- = \mathbf{u}_{rel}^+ \qquad (2.17)$$

Here ϵ is the coefficient of restitution, and its value is between 0 and 1. This law

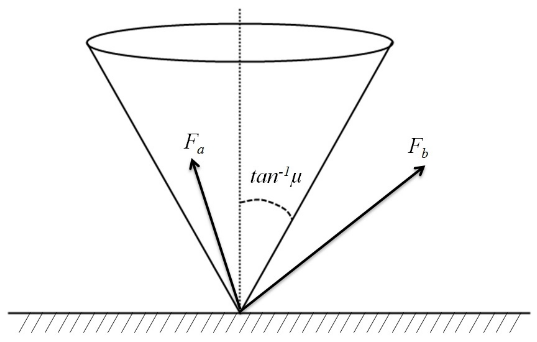


Figure 2.1: Friction cone model for switching switch between static friction and kinetic friction. If the contact force are inside the cone (e.g. F_a), we compute the impulse with static friction. Otherwise with kinetic friction (e.g. F_b).

is correct for the relative velocity in the contact normal direction.

With this in mind, we now consider restitution equation in static friction and kinetic friction cases. We split the relative velocity to tangential and normal directions, $\mathbf{u}_{rel} = u_{rel\_t}\mathbf{t} + u_{rel\_n}\mathbf{n}$. Here u_{rel_t} and u_{rel_n} are scalars; $\mathbf{t}$ and $\mathbf{n}$ are the normalized tangential and normal directions, respectively. For the static friction case, $u^+_{rel\_t}$ should be zero. Reformulating (2.17) by considering this condition, we can have

$$\begin{aligned}
-\epsilon u^-_{rel\_n}\mathbf{n} &= u^+_{rel_n}\mathbf{n} \\
&= \mathbf{u}^+_{rel} \\
&= \mathbf{u}^-_{rel} + K_T\mathbf{j} \qquad (2.18)
\end{aligned}$$

Then $\mathbf{j}$ can be solved as

$$\mathbf{j} = K_T^{-1}(-\mathbf{u}^-_{rel} - \epsilon\mathbf{u}^-_{rel\_n}N) \qquad (2.19)$$

Finally, let's consider the switch between static friction and kinetic friction. We first check this impulse value of (2.19) and compute its normal component and tangential component. We then use a friction cone model to determine which mode it should be. As illustrated in Figure 2.1, a friction cone model generates a cone region based on the friction coefficient. If the contact forces are inside the cone, we compute the impulse with static friction; otherwise we compute the impulse with kinetic friction. Therefore, given a friction coefficient, μ, if $j_t \geq \mu j_n$, we should switch to a kinetic friction case. In the kinetic friction case, we let $j_t = \mu j_n$ by assuming the objects get the largest friction force. So the impulse becomes

$$\begin{aligned}\mathbf{j} &= j_n\mathbf{n} - j_t\mathbf{t} \\ &= j_n\mathbf{n} - \mu j_n\mathbf{t} \\ &= j_n(\mathbf{n} - \mu\mathbf{t}) \\ &= j_n\mathbf{k} \qquad (2.20)\end{aligned}$$

Then, taking a dot of (2.17) with $\mathbf{n}$ on both sides, we can have

$$\begin{aligned}-\epsilon\mathbf{u}_{rel}^{-}\mathbf{n} &= \mathbf{u}_{rel}^{+}\mathbf{n} \qquad (2.21) \\ -\epsilon u_{rel\_n}^{-} &= (\mathbf{u}_{rel}^{-} + K_T\mathbf{j})\mathbf{n} \\ &= u_{rel\_n}^{-} + \mathbf{n}^T * K_T\mathbf{j} \qquad (2.22)\end{aligned}$$

j_n can be solved by substituting $\mathbf{j}$ in (2.22) with (2.20)

$$\begin{aligned}-\epsilon u_{rel\_n}^{-} &= u_{rel\_n}^{-} + \mathbf{n}^T * K_T j_n\mathbf{k} \qquad (2.23) \\ j_n &= \frac{-(1+\epsilon)u_{rel\_n}^{-}}{\mathbf{n}^T * K_T\mathbf{k}} \qquad (2.24)\end{aligned}$$

Once j_n is solved, $j_t = \mu j_n$.

2.2 Aggregate Problem of Rigid Body Simulation

In rigid body simulation, some problems emerge when objects start to aggregate. First, to make objects able to aggregate together, friction modeling is necessary. Second, the penetration problem is more severe in this situation than other situations. The previous two points sound more like problems of core simulation techniques while the third problem, which arises from the usage of object aggregations, is the demand of artistic control for aggregation.

Friction is fundamental to aggregation. Without friction, objects can not pile up. However, modeling friction is not easy. If we use an analytic method, considering friction makes the resting contact not easy to solve. Many approaches proposed using either approximated models or ingenious solvers to compute friction forces more stably and efficiently[37][38].

Instead, if impulse-based methods are used, the penetration problem is much more severe when objects are piling up. For an object inside a pile, if it is penetrating with another object, it also could be pushed in the penetration direction by other objects around its nearby area. Penetrations are not easy to recover from this situation. In addition, for an impulse based method, since we solve the resting contacts in a local scheme (one by one) the sequence to solve them is very critical. While objects are stacked together, resolving a penetration could cause more penetrations under this scheme. It does not guarantee all of the resting contacts will be resolved. Some stop-based approaches are proposed to mitigate this problem[63][24]. Basically, they stop the object when an object's energy or velocity is low.

Even with the improved modeling techniques of friction or contact, we still cannot just rely on friction to control the aggregation. This is simply because there is no direct mapping between the appearance of an aggregation and friction. Modeling massive aggregated objects by hand takes up a great deal of time so people are looking for easy and intuitive methods to control the look-and-feel of aggregations.

2.3 Background: Static Equilibrium Analysis

Static equilibrium analysis is a method to identify whether a stacking structure is steady or not. For a stable stacking structure, it satisfies equilibrium constraints, compression constraints, and friction constraints [44, 27]. Although it has been mainly used in structural and mechanical engineering, recently Whiting et al. [71] utilized this idea to automatically select parameters for procedurally creating masonry buildings with a feasible look and feel. Our method is inspired by their work. In the following, we give an overview of static equilibrium analysis. We also refer readers to Whiting's PhD dissertation [73] for understanding more detail.

Before applying analysis, we must find all contact points between objects. If objects are in contact across an edge or surface (rather than at just a point), the points are chosen from the vertices of the boundary of the overlapping area.

$$\mathbf{A}_{eq} \cdot \mathbf{f} + \mathbf{w} = 0 \tag{2.25}$$

$$\mathbf{A}_{fr} \cdot \mathbf{f} \leq 0 \tag{2.26}$$

$$\mathbf{f}_n^i \geq 0, \quad \forall i \in \text{sampled contact points} \tag{2.27}$$

Equation (1) is the equilibrium equation, which states that the sum of all forces and torques on all contact points of the structure should be zero. In equation (1), $\mathbf{w}$ is a vector containing the weight of each object, $\mathbf{f}$ (the unknown) is the vector of contact forces, and $\mathbf{A}_{eq}$ is the coefficient matrix of equilibrium equations.

Equation (2) states the friction constraints. Note that in order to linearize the system, a friction pyramid model is used to substitute for the friction cone model:

$$|f_{t_1}^i|, |f_{t_2}^i| \leq \mu_s f_n^i, \quad \forall i \in \text{sampled contact points} \tag{2.28}$$

Here μ_s is the coefficient of static friction.

Equation (3) is the compression only constraint. This constraint ensures there is no tensile strength due to the property of rigid body stacking. Given a set of contact point samplings of the structure, we can create a system of linear inequalities based on these three equations. If $\mathbf{f}$ is solvable in this system, the whole structure stands stably. This system can be solved by linear programming if there exists a solution [44].

2.4 Related Studies

This section discusses several previous studies that have been used to improve performance and controllability of dynamic simulation in Computer Graphics.

2.4.1 The Concept of Simplified Simulation

Many approaches are designed to trade off simulation accuracy and speed, allowing simulations to vary in accuracy as needed.

Level of detail techniques are now a well-understood area [45], and many of the principles of LOD carry forward, with some modification, to SLOD. The goal of simulation level of detail(SLOD) is to provide a tradeoff between quality and performance for simulations. SLOD was introduced to computer graphics through the work of Berka [8] and Carlson and Hodgins [9]. The applications of SLOD are varied, including dynamics systems [9, 57], particle systems [56], hair animations [70], and plant motions [6, 19]. To achieve maximum scalability, SLOD usually involves view culling techniques [9, 12].

In addition, Chenney and Forsyth [11] worked on ways to simplify simulations, including early work on certain types of simulations being replaced by statistical models.

There has also been a push toward simplifying simulation by modeling modes of response. James developed some of the key techniques related to modeling of deformations using dimensionality reduction [34]. This work was later followed up by Treuille et al. [67], who applied it to fluid simulation. This sense of reducing computation in real-time by precomputing animation and extracting a simplified model of behavior has motivated the approach we explore here.

Barzel et al. produced one of the earliest papers arguing for the need for plausible motion over precise motion [5]. While their focus was somewhat different, in part arguing that less precision sometimes leads to *greater* plausibility, this basic theme was echoed by several later papers. O'Sullivan et al. [57, 58] have done a significant amount of work in coupling such perceptual studies with simulation.

2.4.2 Simplified Collision

Collision detection is a well-explored field. Rather than reviewing it extensively here, we suggest referring to one of the many survey papers (e.g. [36]) or books (e.g. [1]) that discussion collision detection.

Simplified collision detection is a very common approach used in the real-time graphics community. Typically, in order to achieve the frame rates desired, object collisions are computed with far simpler geometry. A bounding volume hierarchy may be used, but evaluated only to a particular depth, in order to maintain a good frame rate [32].

However, in the cases where such approaches are used, the collision detection is usually all that is desired (e.g. to find if a character has hit some other object). For rigid body simulation, more is needed - the point(s) of collision and from that the response must be calculated. To achieve plausible collision response, the simplified collision detection approach is not sufficient.

Although the principles may have wider applicability, our work focuses on rigid body simulation. For deformable objects, other approaches for speeding up simulations (e.g. via conservative bounding spheres) have been explored [35].

Work on simplified collision response is far more limited. One approach has been to use particle systems instead of rigid bodies [2, 33], but here one must maintain several particle-particle linkages, which can be problematic.

Collisions have also been approximated for control, rather than for simulation speed. Popović et al. [61] modified collision responses in order to control simulation behavior, and Twigg and James[69] used modified collisions to "reverse" simulations from a solved steady state.

2.4.3 Realtime Rigid Body Simulation on GPU

Graphics processing units (GPUs) provide great ability to handle massively parallel computations. Recently, many works related to rigid body simulation utilize the power of the GPU to speed up the computation.

Many GPU-based collision detection algorithms have been proposed[64, 62]. For broad phase, spatial subdivision[23] and sweep-and-prune[42] algorithms are able to be parallelized and implemented on the GPU. Moreover, Bounding Volume Hierarchies can be constructed by GPU-based parallel algorithms[39]. These methods clearly speed up performance. Liu et al.'s [42] method is specifically designed for multi objects collision detection. It can handle broad phase collision detection with

1 million spheres at around 10 Hz.

Also, the solvers for collision response can be implemented on GPU. It is relatively easy to parallelize penalty and impulse methods on the GPU since they handle contacts one by one. So, most of the research on GPU accelerated rigid body dynamics aims to the development of LCP solvers to solve a large number of contacts in parallel. Harada [26]introduced a parallel conjugate gradient method for solving the contact constraints. Tonge et al. [65] proposed an efficient Jacobi-based LCP solver. A more comprehensive literature about using GPU solvers for rigid body simulations can be found in Bender et al.'s report[7].

2.4.4 Contact and Friction Stabilization

Modeling contact and friction accurately helps objects rest stably in simulations. Solving for contact and friction separately, which most solvers do, leads to stability issues. Kaufman et al. [38] propose a coupled model for solving contact and friction together and demonstrate that the method is able to simulate structures that need a lot of friction for support.

Temporarily stopping some objects is another way to stabilize structures. Mattikalli et al. [48] proposed a method for stably assembling objects under gravity. The method finds critical positions and pins those positions during an assembly task. Guendelman et al. [24] introduced the idea of shock propagation on a impulse-based contact solver to improve stability of object stacking. Erleben[20] extended the idea to an analytical contact solver. Most of these works increase stability from the fundamental formulations of contact and friction, while our work tries to achieve stability by utilizing static equilibrium analysis.

Recently, Whiting et al. [71, 72] also utilized static equilibrium analysis to automatically select parameters for procedurally creating masonry buildings with a feasible look and feel. But their work focused on improving the structural stability by exploring the variations of the geometry shapes.

2.4.5 Directable Simulation Control

The idea of directable simulation control in a physically-based simulation system is to adjust the simulation process in a way that matches a user's direction while maintaining visual plausibility.

Many approaches can achieve this idea such as matching keyframes or examples[49], matching simulations in different levels of detail[31, 53] or using a time-reverse simulation scheme[69].

Directable simulation control has been explored in most areas of simulation, including rigid body simulations[10, 61, 60, 68, 69], deformable object simulations[47], and fluid simulations[49, 21, 55, 54, 31, 53].

For multibody control, Popović et al. [61, 60] used optimization techniques to search for a solution between the initial and the end configuration specified by the user. This technique works very well for simulations involving a small number of objects, but it has performance issues for large scale simulations. Twigg and James[68] introduced a selection-based approach to achieve simulation control. They provide an interface that users can easily review and select desired results from a precomputed example database. Time-reverse simulation is another interesting technique proposed by Twigg and James[69]. Although it starts the simulation from the ending configuration and steps the simulation backward (t, $t - \Delta t$, $t - 2\Delta t$), it maintains the correct look and feel of the simulation when we play the result forward (0, Δt, $2\Delta t$). However, to handle complex stacking situations, it needs special treatment, and the sliding, rolling and bouncing motions are limited to the top of the stacking structure.

2.4.6 Object Aggregation Modeling

Modeling massive aggregated objects by hand takes up a great deal of time. Rather than trying to model the individual contact and stacking forces and have a pile result, a few papers aim to model the pile itself and make the objects adapt to it.

As objects fall, a conical hill tends to be formed little by little. An interesting fact is that no matter how many of the same objects are piled up, there exists a maximal stable slope of that pile. The angle of the slope is called the angle of repose [75, 43]. This concept has been used previously in graphics to generate realistic models of material (such as snow) in a final position [22], but has not been used directly in simulation until now.

The maximal angle of repose is related to several factors: the size of objects, the shape of objects, and the material friction. In order to generate piles "correctly" within a simulation, simulation of the contact frictional forces is necessary. This tends to be a much more complex and time-consuming approach, and so often the models of static and dynamic friction are replaced with simplifications, e.g. frictional components within an impulse response. The net result is that while a typical simulation will support some piling due to object geometry and the limited approximations of friction, the piles that result are often shallower (i.e. a lower angle of repose) than those that might be desired.

In addition, several techniques have been proposed to generate aggregations. Ma et al. [46] proposed an example-based approach to synthesize repetitive elements across a user specified domain. Peytavie et al. [59] introduced an aperiodic tiling technique to generate rock piles procedurally. Cho et al. [13] used an upside down mesh as a container for catching objects dropped into it, then turned the container upside down again when it finished. Fearing[22] took the concept of angle of repose to generate realistic snow piles. These methods are designed to generate static models or offer only limited control over shapes. Modeling massive aggregated objects for dynamics use is still a challenging problem.

3. STATISTICAL SIMULATION OF RIGID BODIES

3.1 Introduction

The most computationally significant portions of rigid body simulation are in the detection and response to collisions. In a typical system, this process involves three stages: collision detection, determination, and response. Collision detection involves finding whether or not a collision has occurred for a rigid object. Collision determination, which is sometimes combined with collision detection or response, involves finding the exact point at which a collision occurs. Collision response involves computing the response of the object to the collision. A typical method for doing this is to compute an impulse that is applied to the object, changing its linear and angular momentum. Usually, collision detection and determination occupy the largest fraction of time in the simulation. Collision response is a relatively fast calculation, but relies on accurate collision determination in order to produce realistic results.

In this chapter, we describe a statistical method to simplify four important steps of the rigid body simulations: collision detection, collision determination, collision response, and resting. Our approach modifies each of these steps by using simplified calculations based on a statistical "signature" precomputed for the object. Collision detection and determination are replaced by a simple collision detection computation based on a sphere, where the radius varies based on statistical analysis of typical collisions for the object. Collision determination is eliminated. While this alone provides most of our performance improvement, it poses challenges for collision response, since we no longer have an accurate point of collision. For collision response, we use statistical data to directly change linear and angular momentum, producing a plausible response. Furthermore, we directly modify the objects as they approach a rest state, in order to ensure that objects come to rest in configurations matching those commonly encountered statistically.

While this approach does lose some fidelity, the statistical properties of the rigid body motions are maintained. Further, these simplifications provide significant increases in simulation speed. This makes our approach highly suitable for any situation in which a tradeoff of fidelity for performance is desirable.

3.1.1 Main Results

The result of this chapter shows that a statistically-based collision approach can provide plausible results. We describe how such a system can be implemented. In particular, we discuss below:

- how a statistical signature can be precomputed by analyzing several representative applications (Sec. 3.2),
- how the statistical signature can be used to simplify the collision detection, determination and response calculations,
- how the particular case of rigid objects coming to rest (on a planar surface) is handled.
- how to save the storage of the collision response signature by fitting the data with B-Spline hypersurfaces to save storage (Sec. 3.3).

3.2 Methodology

3.2.1 Overview

The fundamental idea of our statistically-based rigid body simulation is to approximate the collision model of a rigid object by an dynamic sphere, which will boost the computation performance. However, the collision response behavior of a simplified sphere collision model is different from the behavior of the original collision model. To more accurately approximate the collision response behavior of the original collision model, we develop a statistically-based approach to blur the error caused by the simplified sphere model, including how to determine the size of the dynamic sphere and how to synthesize plausible collision response velocities.

Our approach has two stages. In the first stage, which is precomputed, we run several simulations for individual objects from a variety of collision situations. We use these results to collect a statistical "*signature*" for that object's response to collisions. In the second stage, we use the statistical signature during run-time to quickly compute collisions and responses of that object type. When objects are approaching a rest condition, we modify our approach slightly in order to ensure the object ends in a valid rest state.

3.2.2 Reduction of Sampling Space for Determining Collision Response

As mentioned above, our method has to sample a series of collision situations, compute the response velocities from those collisions, collect these results, and utilize them in the run-time stage. We will collect this data by examining a particular, simple, scenario in which we have only an object and a horizontal plane, and the object is shot from the air to intersect the plane. In this scenario, a combination of the object's orientation ($\mathbf{q}$), linear velocity ($\mathbf{v}$), and angular velocity ($\mathbf{w}$) at the time of collision is called a "collision configuration." Note that the collision configuration is 9-dimensional, and that it must be determined at the point of collision. For the scenario we deal with, the collision configuration is sufficient (assuming fixed parameters such as mass, elasticity, etc.) to determine exactly what the collision response should be.

We could sample collision responses across this entire 9D space. However, there are two main issues with doing so. First, determining the configuration requires knowing the exact point of collision. Precise collision determination is the most time-consuming portion of the collision detection-determination-response process. In the interest of speed, we would like to simplify this collision determination, and thus may not have a collision configuration at the precise point of collision. Second, the 9 dimensional data presents too large of a space to sample effectively. For both time (to sample this space) and space (to store the sampled data) reasons, we need to work with a smaller dimensional data set.

The original 9D sampling space covers the magnitude of $\mathbf{v}$ and $\mathbf{w}$, the direction of $\mathbf{v}$ and $\mathbf{w}$, and the orientation $\mathbf{q}$ of the object. We first reduce the sampling space to a 6D space by giving up the 3 dimensions of $\mathbf{q}$. Knowing the orientation $\mathbf{q}$ requires precise collision determination, and as mentioned above, we do not want to spend the time for such precise determination in our statistically-based simulation. We instead will treat the object as a sphere (as we discuss below), and will ignore the precise orientation. During the sampling phase, we will randomly select orientations and collect statistical data for the range of possible orientations.

We can then go a step further to reduce the sampling space to 4D. If we sample the magnitude of $\mathbf{v}$ and $\mathbf{w}$ and the direction of $\mathbf{v}$ and $\mathbf{w}$ just on a plane which is perpendicular to the horizontal plane, the sampling space can be reduced from 6D to 4D (i.e. two scales for the magnitudes and two angles for the directions). Then, by transforming the sampling results from the horizontal plane, it is able to approximately cover the original 6D sampling space. The detail of how to sample collision configurations in this reduced space will be discussed in section 3.2.3.2.

As a result, we will collect statistical data that spans a 4D space, though the test scenarios will be sampled from the entire 9D space.

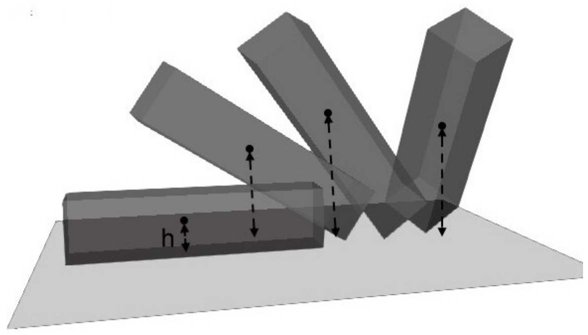


Figure 3.1: Illustrating the collision height. Height measures the distance of the center of mass from the plane an object is colliding against.

3.2.3 Determining a Signature

We consider an object's *signature* to be a collection of statistical data regarding its behavior under collision configurations. We determine a signature by running numerous tests from various starting configurations, measuring the values we are looking for directly, and then extracting both the mean, μ, and standard deviation, σ, for those statistics. There are three main parts to the signature, which we discuss here.

3.2.3.1 Determining Sphere Radius

Perhaps the most important part of the signature is the radius of the sphere to be used for collision. During our simplified simulation, we will replace the object geometry with a simplified sphere geometry. We must determine the appropriate radius of sphere to use. To do this, we determine height, the distance of an object's center of mass from a plane at the time of collision. Figure 3.1 illustrates the height measurement for a box.

Height can vary significantly, depending on the dimensions of the object. Furthermore, height is correlated very closely with angular velocity, $\mathbf{w}$. Considering

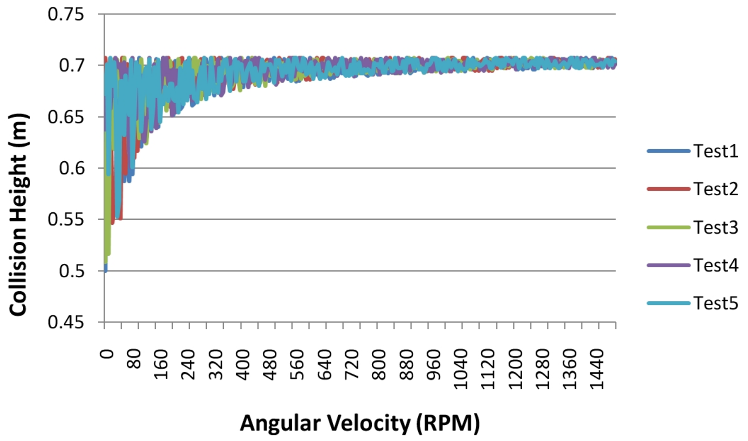


Figure 3.2: Graph showing the range of collision heights for a 1×1 square falling against a line. Angular velocity is measured relative to a fixed linear velocity. As the angular velocity increases, the collision height approaches $\sqrt{2}/2$.

the box example from Figure 3.1, the minimum height shown can only occur when the box collides exactly along its large face. Generally this requires that the box be falling straight down with no angular velocity. As $|\mathbf{w}|$ increases, it is not possible for the collision to occur at this height, since the object will rotate as it approaches the plane. One of the corners or edges of the box will collide before the face can collide.

This is illustrated in Figure 3.2 for a 2D square (side length = 1). The graph shows the range of recorded collision heights recorded from a series of experiments with random starting heights at a variety of angular velocities. With no angular rotation, the height at collision may be anywhere from 0.5 to $\sqrt{2}/2$. As angular rotation increases, the height approaches $\sqrt{2}/2$. For a given angular velocity, then, we can compute the mean (μ) and standard deviation (σ) of the heights over a variety of simulations. We store this in a table. While this simple case could have been computed analytically, it illustrates the process used for more complex shapes. Further, while this may seem obvious for the simple $2D$ square, similar behavior is seen for more complex objects in $3D$.

We call the height at collision h_{coll}. To collect h_{coll} data, we run several trials (notated as n in Equation 3.1) for a given object. We fix the linear velocity, $\mathbf{v}$, and use a variety of random angular velocities, $\mathbf{w}$. Note that $\mathbf{w}$ must be recorded relative to the magnitude of $\mathbf{v}$ (i.e. it is not the absolute rotation, but rather the rotation relative to linear velocity that matters). As a result, this table is indexed by $|\mathbf{w}|/|\mathbf{v}|$, notated as r in Equation 3.1. Given a range of $|\mathbf{w}|/|\mathbf{v}|$ values, extracting μ for that range is straightforward:

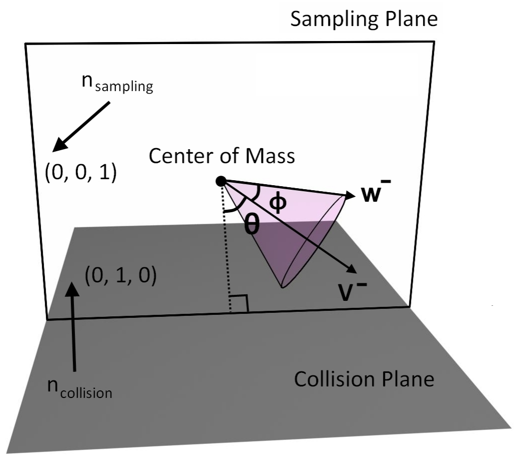
$$\mu^{h_{coll}}(r) = \frac{\sum_{i=1}^{n} h_{coll}^{i}(r)}{n}. \tag{3.1}$$

We store $\mu^{h_{coll}}$ in the collision height table, which can be queried by the value of $|\mathbf{w}|/|\mathbf{v}|$.

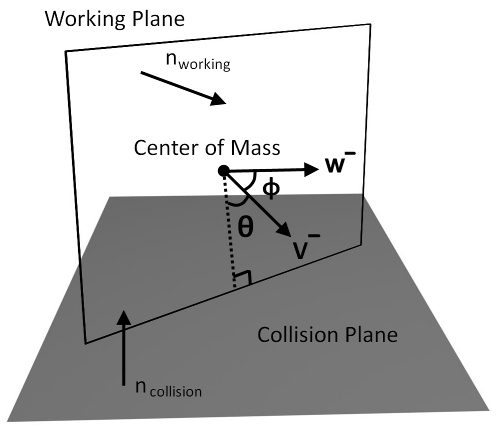
As can be seen in Figure 3.2, the height tends to vary much more for lower values of $|\mathbf{w}|$ than for larger ones. Thus, to assure that we capture the variation in μ accurately, we use a non-uniform sampling over the range of $|\mathbf{w}|$ values. Specifically, we sample $|\mathbf{w}|$ more closely at lower values, and much more sparsely at higher values. Later, we can reconstruct μ for any $|\mathbf{w}|$ value by interpolating between the nearest samples.

3.2.3.2 Determining Collision Response

The second key part of the signature is the characterization of the collision response. Collision response characterizes the result of a collision in terms of the change in $\mathbf{v}$ and $\mathbf{w}$ due to the collision. Again, we will perform a series of trials, and determine μ and σ for various indexed values in the table. In the sampling environment, we have two planes. The first one is a collision plane, which is a geometric object for colliding with the dropped object. The second plane is called sampling plane, which



a



b

Figure 3.3: We use a sampling plane (a) to collect data. $\mathbf{v}^-$ is the linear velocity before collision, by definition in the sampling plane, moving at an angle θ to the plane normal. $\mathbf{w}^-$ is the vector expressing angular velocity (the direction giving axis of rotation, and the magnitude the absolute velocity). $\mathbf{w}^-$ forms an angle ϕ with $\mathbf{v}^-$. Note that $\mathbf{w}^-$ does not necessarily stay on the sampling plane. To retrieve the sampling data during simulations, we first locate a working plane (b), on which the $\mathbf{v}^-$ and the collision normal stay. Also, we project $\mathbf{w}^-$ to the working plane. Finally we compute θ and ϕ to and use them along with $\mathbf{v}^-$ and $\mathbf{w}^-$ to query the response velocities.

is served as a reference plane for systematically sampling the collision scenarios. To control the collision scenarios, no gravity or external forces are applied.

Again, as mentioned in Sec. 3.2.2, the sampling space of collision scenarios is reduced to a 4D space, so we will collect data in a table indexed by four different values. These values are illustrated in Figure 3.3a. Specifically, a sampled collision configuration(χ) includes

- θ, the angle between $\mathbf{v}^-$ and the inverse direction of the plane normal.
- ϕ, the angle between $\mathbf{v}^-$ and $\mathbf{w}^-$.
- $|\mathbf{v}^-|$, the magnitude of linear velocity.

- $|\mathbf{w}^-|$, the magnitude of angular velocity.

First, we force $\mathbf{v}^-$ to stick on the sampling plane. However, this restriction is not necessary for $\mathbf{w}^-$. Given a ϕ, the possible directions of $\mathbf{w}^-$ form a cone as shown in Figure 3.3a). We will randomly pick one possible direction for each trial. Note that $\mathbf{v}^-$ is set so the object moves toward a plane.

Again, we do not have to sample these values uniformly, but can adjust the sampling to better capture wider variations in the measured statistics.

We use an impulse-based simulation with precise collision detection to determine the collision location and response. From this, we compute three response values:

- $\mathbf{v}^+_{rigid}$, the linear velocity after collision.
- $\mathbf{w}^+_{rigid}$, the angular velocity after collision
- $\mathbf{v}^+_{particle}$, the resulting linear velocity we would expect if the object were simply a sphere (or particle) rather than a rigid object.

$$\mu^{\mathbf{v}_{rectified}}(\chi) = \frac{\sum_{i=1}^{n}(\mathbf{v}^+_{rigid^i})(\chi) - (\mathbf{v}^+_{particle^i})(\chi)}{n} \tag{3.2}$$

We collect and store data in two tables (each indexed in the four dimensions listed earlier). One table (the $\mathbf{w}$ table) contains μ and σ values for the angular velocity ($\mathbf{w}^+_{rigid}$). The other table (the $\mathbf{v}$ table) contains μ and σ values for the *difference* in linear velocity from a particle response. Equation 3.2 shows how to compute the μ of the rectified linear velocity. While typically the μ value for that table is near 0 (i.e. on average, the response is similar to a particle bouncing), the σ value certainly is not. Note that this is one of the key areas where our statistical signature provides additional information that could not be obtained by simply treating the whole object as a sphere.

3.2.3.3 Determining Resting Configurations

The third aspect of the signature that we collect is information about the typical resting conditions (on a plane) for an object. As an object comes to rest, there are typically only a few stable configurations it can end up in. Mirtich [50] previously explored the idea of estimating common resting configurations (or in his terms, "pose statistics"), however most previous work has focused on finding such stable configurations automatically from the geometry (and this is not always an easy operation). For some models, even though some configurations might technically be stable, those configurations rarely occur in practice (e.g. if small forces would tend to upset the stable configuration). An example would be a long skinny box: though it could end up resting on one end, it is far more likely to end on one of its larger sides. Thus, we again measure resting conditions using a statistical approach.

We perform several simulations, in a variety of starting configurations (differing orientations and velocities), but apply gravity to the simulation. Again using an impulse-based simulator, we simulate the object precisely until it reaches a resting configuration (momentum falls below a threshold). We collect two pieces of data:

- The height of the center of mass when the object is at rest, h_{rest}). Note that this will often be much smaller than the μ of h_{coll} at $\mathbf{w} = 0$.
- The rest orientation, $\mathbf{n}_{rest}$, giving the plane normal expressed in the object's local coordinate system.

Notice that each $\mathbf{n}_{rest}$ should correspond to a unique h_{rest}. Also, $\mathbf{n}_{rest}$ represents the "up" direction for the object in that rest configuration; rotating around that vector should still give a stable rest configuration.

In this case, we do not compute μ and σ values. Rather, we run several trials, and determine the $(\mathbf{n}_{rest}, h_{rest})$ values. For those that occur often enough (e.g. more than 5% of the time), we store this as a “valid” rest configuration within the object signature.

3.2.4 Using Signature Data to Approximate Collisions

Given the signature data for a model, we can use it at run-time to simplify our collision calculations. We will describe our basic approach here, and discuss how we handle resting conditions in section 3.2.5.

3.2.4.1 Determining Object Collision

At run-time, we replace the geometry of the object with a simple sphere, and use this to test for collisions. The same sphere is used for both object-plane and object-object collisions. To determine the radius of the sphere to use (spheres are always centered on the center of mass), we compute the relative magnitude of the angular velocity to the linear velocity, i.e. $|\mathbf{w}|/|\mathbf{v}|$. We use this to index into the table (both interpolating neighboring values and simply using the nearest value work well) of h_{coll}. This table gives us the mean value, μ, of the collision distance, and we use that for the radius of the sphere. Note that the sphere radius may change from timestep to timestep; since this change is usually minor, we can limit the frequency of sphere size updates if desired (we have not done this in our implementation results presented below).

When an object is found to collide with another object or with a plane (polygon), we apply the collision response calculation described next.

Note that this approach is focused on the narrow phase of collision detection. It can be easily augmented by standard broad-phase collision detection speedups, such as a sweep-and-prune operation or spatial subdivision. It is best to use a sphere of the maximum size possibly needed (i.e. a bounding sphere) for the broad-phase calculations, since otherwise temporal coherency may be lost due to updates of sphere sizes.

3.2.4.2 Determining Object Response

To calculate collision response, we make use of the $\mathbf{v}$ and $\mathbf{w}$ tables stored in the signature to determine a statistically plausible collision response. Refer to figure 3.3b for some of the notation. The specific steps are as follows:

1. *Calculate the collision plane normal.* If the object collides with a plane, that plane is the collision plane. If the collision is with another object (represented by a sphere), the plane is the separating plane between the two spheres at the point of collision (perpendicular to the line connecting the sphere centers). Note that only the collision plane normal, $\mathbf{n}_{collision}$, is needed in our scheme (we do not need the actual plane or the penetration depth of the object pair).

2. *Calculate the working plane normal.* We define the working plane to be the plane spanned by the collision plane normal and the object's linear velocity; its normal is $\mathbf{n}_{working}$.

3. *Compute a rotation matrix from the canonical system to the working system.* The data stored in the $\mathbf{v}$ and $\mathbf{w}$ tables is computed relative to the canonical system used for sampling (see Figure 3.3a). We must be able to convert that data to the system (defined by the collision plane and working plane normals) encountered at runtime (see Figure 3.3b). This rotation matrix (formed from the collision and working plane normals) is applied to the results of the later steps (we do not explicitly state this later).

4. *Compute the current configuration.* For the working system, we need to compute the values of θ, ϕ, $|\mathbf{v}^-|$, and $|\mathbf{w}^-|$. These are computed as in the signature

determination stage for collision with a plane. For collision with another object, the velocity is determined relative to the other object (i.e. as if that object were fixed).

5. *Retrieve statistical data.* The configuration data is used to retrieve values $(\mu_{\mathbf{v}}, \sigma_{\mathbf{v}})$ and $(\mu_{\mathbf{w}}, \sigma_{\mathbf{w}})$ from the $\mathbf{v}$ and $\mathbf{w}$ tables.

6. *Determine the new $\boldsymbol{w}^{+}$.* We form a Gaussian distribution based on $\mu_{\mathbf{w}}$ and $\sigma_{\mathbf{w}}$, and we then generate a random sample from that distribution.

7. *Determine the new* $\mathbf{v}^{+}$. We first calculate a linear velocity response, $\mathbf{v}^{+}_{particle}$ based on a simple calculation as if the object were a particle. Then, similar to $\mathbf{w}$, we generate a random value, $\mathbf{v}^{+}_{rectified}$ based on $\mu_{\mathbf{v}}$ and $\sigma_{\mathbf{v}}$. The final value is then determined as $\mathbf{v}^{+} = \mathbf{v}^{+}_{particle} + \mathbf{v}^{+}_{rectified}$.

8. *Update* $\mathbf{v}$ *and* $\mathbf{w}$. The linear and angular velocity are simply replaced by the new $\mathbf{v}^{+}$ and $\mathbf{w}^{+}$.

For object-object collision response, it would be impractical to sample and store data from all possible collision configuration pairs. Instead, we approximate these collisions using the object-plane response data. For each collision pair, we first compute the collision plane (working plane) between the spheres approximating each object. We then apply the object-plane collision response for the two objects as if they were colliding with that plane. The two object responses are computed independently.

The end result is that the object responds to the collision by receiving a new linear and angular velocity that fall within the statistical range of observed responses for similar initial states.

3.2.5 Achieving Resting States

The above approach works well for most dynamic scenes, but has a serious deficiency when objects come to rest. In particular, there are no restrictions on the final orientation of the object, so objects may come to rest in awkward and invalid states. Although people will often not notice this until it is pointed out, and it may make no difference depending on level of detail required for the simulation (e.g. it may be so far away this makes no difference), there will be cases where we wish to ensure that the resting states of the objects are valid. We do this by applying "resting conditions" that explicitly move the object to a valid rest states.

We ignore resting conditions until we detect that an object has sufficiently low linear and angular momentum, within a certain distance from a horizontal plane. For objects that come to rest in other situations (e.g. if the objects are piled up on top of others), we do not do anything special; once the object has come to a near-still state, we set its velocities to zero and consider it at rest. In such cases, objects do not have a limited well-defined set of rest poses, anyway, so the need for applying our resting conditions is much less.

For the objects that we wish to apply resting conditions to, we first determine the nearest $\mathbf{n}_{rest}$ value from the signature, given the current configuration (we choose the one requiring the smallest angular rotation from the current orientation). We then gradually modify the object so that it moves toward this rest state.

One part of the modification is replacement of the sphere radius from the table with the h_{rest} corresponding to the chosen $\mathbf{n}_{rest}$. A sudden replacement may cause the object to suddenly "drop" down (since h_{rest} is usually smaller than h_{coll}), so we transition to this radius gradually over several frames of animation.

The other part of the modification is the motion toward the appropriate orientation. We apply a small rotational impulse to the object at each collision that will move it toward the $\mathbf{n}_{rest}$ orientation. This is done instead of the $\mathbf{w}^+$ calculation described earlier. Since resting conditions are only applied when the object's motion is already small, this change is usually unnoticed. The effect is that the object will gradually bounce toward its stable resting state.

Together, over a series of collisions as the object is coming to rest, it will slowly turn to a valid resting condition. When it finally does come to rest (dropping below an even smaller threshold of linear and angular velocity), it should be in a valid configuration.

3.2.6 Sampling Detail

In our simplified simulation, which we will refer to this system as the *statistical* system, we followed the approach described above. For collecting statistical signature, The total time spent collecting signature data was approximately 6-8 hours. The total storage size for the signature information from a model was approximately 3 MB.

The h_{coll} information was collected at 16 sample values of $|\mathbf{w}|$. For each sampled $|\mathbf{w}|$, we randomly pick 100 directions to collect the collision heights.

For the $\mathbf{v}$ and $\mathbf{w}$ tables, we sampled θ at 7 values, ϕ at 10 values, $|\mathbf{v}^-|$ at 26 values, and $|\mathbf{w}^-|$ at 16 values. We ran 30 trials for each collision configuration for which we extracted μ and σ. This required a total of $873,600$ simulations.

As for resting conditions, we drop a model 100 times to find its resting states. Different models might have different numbers of resting states. For example, we found 4 resting states for the box model we demonstrate, 4 resting states for the bunny model, and 3 for the armadilloman model.

3.3 Fit the Collision Response Signature with B-Spline Hypersurfaces

To save on storage for the table for our signatures, the collision response signatures can be replaced with B-spline hypersurfaces fitted to the original collision response tables. With this substitution, the storage needed for the collision response signatures of one object shape is drastically reduced from 3MB to 5KB. Note that this is an alternative representation of our statistical signatures while the cost of this representation is that it is a little bit slower to query the response velocities by this fitting approach than do it by the table lookup approach, and it will decrease the fidelity of the result slightly.

Our table structure is described as follows. Again, to represent collision response signatures, we use four $4D$ tables to store the mean and standard deviation value of the rectified linear/angular velocities. The table length in each dimension is different. In our case, it is a $7 \times 10 \times 16 \times 26$ table. The data in each cell of the table is a $3D$ vector expressed by Cartesian coordinates.

Before we perform the fitting, we convert the data to spherical coordinates. The reason is we believe spherical coordinates are more natural for presenting direction and magnitude of the data than Cartesian coordinates. Next, for each table, we use a B-Spline hypersurface to approximate the data. Assume the hypersurface function is

$$S(s,t,u,v) = \sum_{i=0}^{n0} \sum_{j=0}^{n1} \sum_{k=0}^{n2} \sum_{l=0}^{n3} N_i(s) N_j(t) N_k(u) N_l(v) P_{ijkl}, \tag{3.3}$$

where P are the control points of the approximated hypersurface, the unknown we want to solve. N is the b-spline basis function. We assume the degree of the basis functions are the same in each dimension. $n0$, $n1$, $n2$, and $n3$ are the number of control points in each dimension.

By minimizing the following square distance term, we can obtain the control

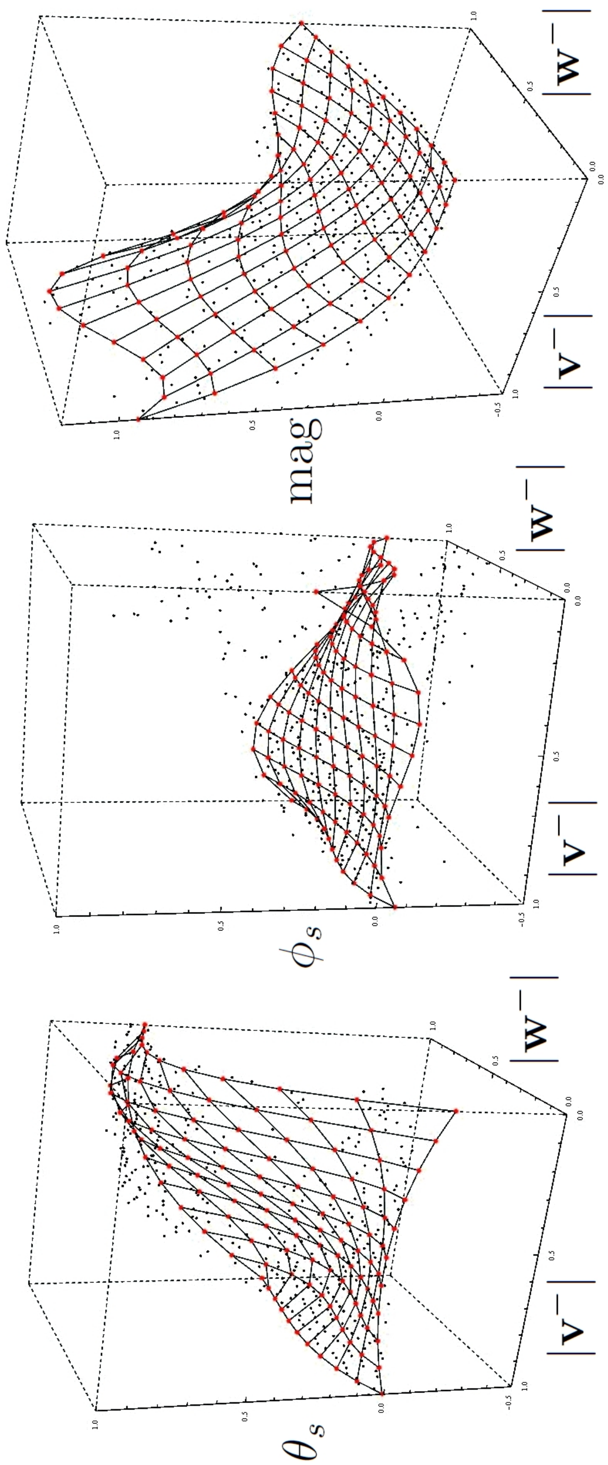


Figure 3.4: The visualization of one instance of our data fitting. We pick a specific θ and ϕ, and the show the fitting results of the rest two dimensions, $|\mathbf{v}^-|$ and $|\mathbf{w}^-|$.

points of an approximated hypersurface fitting to our original data in the sense of least squares.

$$\sum_{i=0}^{n0}\sum_{j=0}^{n1}\sum_{k=0}^{n2}\sum_{l=0}^{n3}|D_{i,j,k,l} - S(s_i, t_j, u_k, v_l)|^2, \tag{3.4}$$

where D are our data points. s_i, t_j, u_k, v_l are the corresponding parameters of $D_{i,j,k,l}$. We need to compute knot values and then parameters for the minimization. We assume the knot values are placed uniformly. Next, we use the centripetal method[40] to compute parameters and generate the coefficient matrix. After that, we can solve this over constrained linear system in the least squares sense to obtain the control points.

Centripetal method is a parameterization approach. We use this approach to compute each data point a parameter value within the parameter domain (usually between 0 and 1). The main characteristic of this method is that the distance between two adjacent data points is computed by $|D_{idx} - D_{idx-1}|^{0.5}$, instead of $|D_{idx} - D_{idx-1}|$. Compared to other parameterization approaches such as uniform and chordal method, the centripetal parameterization can handle sharp turns around the sharp features in the data polygon better.

Similarly, when the simulation is running, given a collision configuration and these control points, we can evaluate a μ and a σ of the rectified linear/angluar velocity and synthesize a new linear/angluar velocity according to this μ and σ.

In our implementation, the degree of basis functions are 3 and the number of control points in each dimension is 4. Figure 3.4 illustrates one instance of our fitting result. The data we pick here is the mean value table of the rectified linear velocity. The θ_s, ϕ_s, and mag are the three spherical coordinates. We specified values for θ and ϕ and show the fitting result of the $|\mathbf{v}^-|$ and $|\mathbf{w}^-|$. Note that the fitting in ϕ_s is not good when the $|\mathbf{v}^-|$ is low. The reason is the azimuthal angle of the response velocity is more random when the $|\mathbf{v}^-|$ is lower.

3.4 Results

In this section, we show the implementations and results of our statistically-based simulator. We will first discuss the implementation details, and then present examples and timing results.

3.4.1 Implementation

We implemented our simulation methods and obtained timing results on a machine with an Intel Core 2 2.66GHz CPU and a NVidia 9800GT graphics card. We first developed a fairly standard impulse-based rigid body simulation system[24]. We will refer to this system below as the *impulse* system. This impulse system is used as the basis for the comparisons and timing evaluations described below, as well as for collecting signature data. The physically-based simulator in our mixed simulator is also based on this one. The impulse system uses SWIFT++ [18] to handle collision detection and determination.

In order to provide a more fair comparison of performance, in our statistically-based simulator (referred to as the *statistical* system), we implemented a bounding-sphere-based sweep-and-prune approach in our simulation. The reason is that SWIFT++ system uses a sweep-and-prune approach in the broad phase to limit the number of collision checks between objects. While this approach makes little difference with small numbers of objects, it is important when dealing with large numbers of objects. We did not implement any further broad-phase algorithms, such as spatial subdivision, in the statistical system since it did not appear that SWIFT++ provided this.

In our mixed simulator (referred as the *mixed* system), we combine the impulse

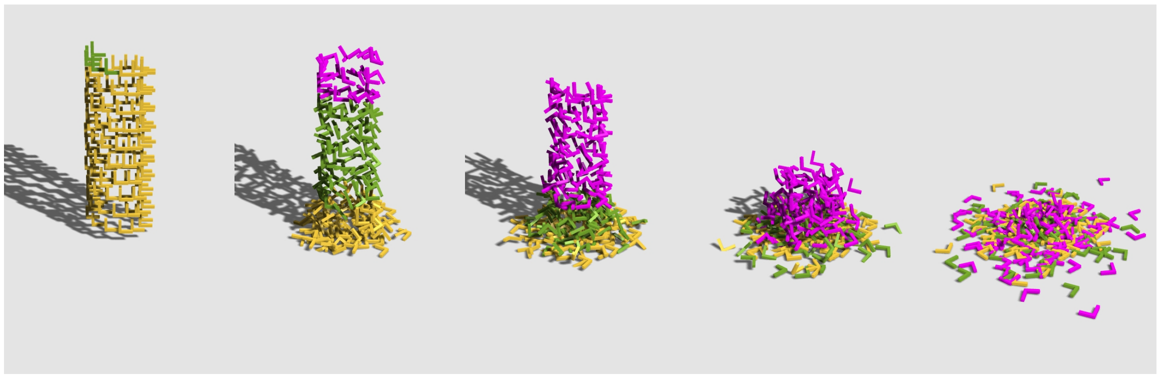


Figure 3.5: Frames from a statistical simulation of 300 L-Shape objects falling on the ground. Standard collision detection and response is replaced by a simplified statistically-based collision detection and response model.

system and statistical system. For collision detection, we use AABB-based sweep-and-prune in broad phase. In the narrow phase, we could use SWIFT++ for impulse cases or sphere-based collision for statistical cases. Note that Swift++ combines both broad and narrow phases, and we had to work around their code to use our own broad phase computation in the mixed simulation; as a result, the combination of our broad phase with Swift++'s narrow phase is less efficient than using Swift++ alone, but the efficiency loss is minor and does not affect any conclusions.

3.4.2 Results: The Statistical System

We present some results of the statistical system in Figures 3.5, 3.6, 3.7, and 3.9. We show results from four basic shapes. The box examples are presented as a "base" case. We use both bunny and armadillo-man shapes to demonstrate the ability to handle more complex objects. Finally, we include examples of an 'L'-shaped object, which we believe should be a "worst case" example for the statistical system (due to the different dimensions and concavity). The overall behavior is such that in informal polls, most were unable to tell which one was "correct" for several simulations (L-

Table 3.1: Performance improvement for the statistical simulation vs. various other simulations. For each simulation, one step refers to the total amount of time taken for a simulation step. Collision is the portion of the time spent in collision detection and determination, and response is the portion of the time spent in collision response. Times are in seconds, and the percentages are the amount of time relative to the impulse response (or the 7-sphere example for 100, 000 objects). We compare three different approaches: a full impulse simulation of the original bunny model (with 2503 vertices and 4698 faces), a full impulse based simulation of the bunny approximated by seven spheres, and a statistical simulation described here.

objects	frames	type	impulse(sec.)		simplified(sec.)		7-sphere(sec.)		statistical(sec.)	
100	420	total	40.3856	100.00%	4.0380	10.00%	0.3378	0.84%	0.1784	0.44%
		one step	0.0393	100.00%	0.0044	11.41%	0.0002	0.73%	0.0001	0.38%
		collision	0.0390	100.00%	0.0043	11.21%	0.0002	0.55%	0.0000	0.25%
		response	0.0002	100.00%	0.0000	42.06%	0.0000	27.47%	0.0000	13.73%
1K	420	total	3214.3925	100.00%	134.1173	4.17%	9.1107	0.28%	3.4784	0.11%
		one step	2.8204	100.00%	0.1527	5.42%	0.0082	0.29%	0.0031	0.11%
		collision	2.8091	100.00%	0.1504	5.36%	0.0067	0.24%	0.0023	0.09%
		response	0.0112	100.00%	0.0022	19.63%	0.0014	12.84%	0.0005	5.06%
10K	420	total	52152.8533	100.00%	5838.5883	11.20%	231.0220	0.44%	70.3196	0.13%
		one step	62.4276	100.00%	4.6671	7.48%	0.2331	0.37%	0.0577	0.09%
		collision	62.1851	100.00%	4.6115	7.42%	0.2036	0.33%	0.0495	0.08%
		response	0.2416	100.00%	0.0546	22.64%	0.0285	11.83%	0.0065	2.70%
100K	420	total					5148.3922	100.00%	2863.3412	55.62%
		one step					3.3960	100.00%	1.4244	41.94%
		collision					2.9964	100.00%	1.3328	44.48%
		response					0.3902	100.00%	0.0743	19.05%

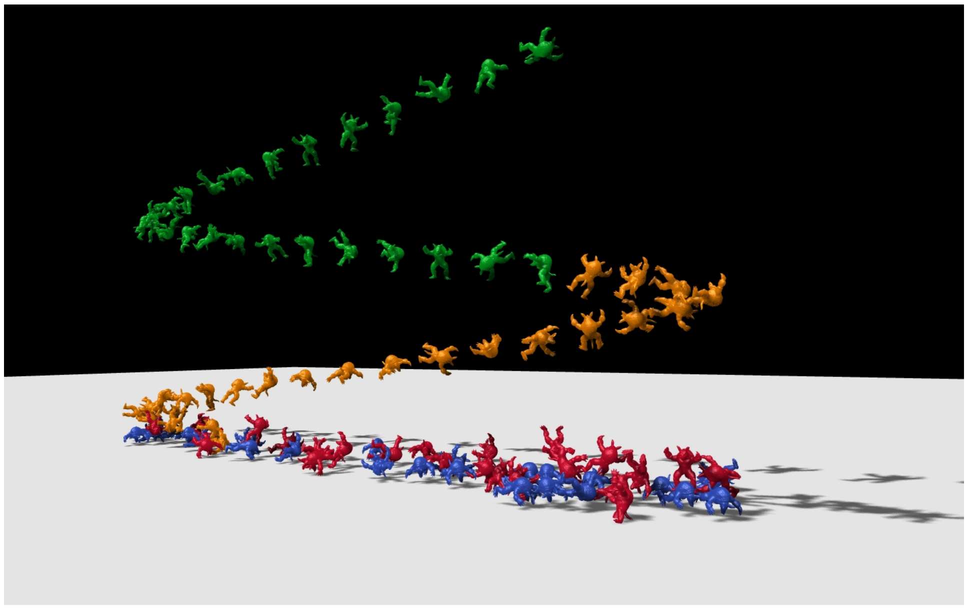


Figure 3.6: Statistical simulation of 100 armadillos.

shape simulations), and sometimes could only determine this when the types of errors to look for were pointed out.

Figure 3.5 shows the result of a statistical simulation of 300 L-Shapes dropped in a cylinder pattern. Figure 3.6 shows a frame from a statistical simulation of 100 armadillos. Figure 3.7 shows a frame from a statistical simulation of $10,000$ bunnies. Figure 3.9 shows a comparison of the impulse and statistical simulations.

A close-up view of the effect of resting conditions is demonstrated in Figure 3.8. Without resting conditions, the bunny model appears in an awkward orientation, and the sphere radius used for collision is too large (the bunny is actually slightly off the floor). Other models may look even worse. With resting conditions, the bunny model appears in a more natural orientation. The collision sphere has a radius appropriate to keep the model in contact with the floor.

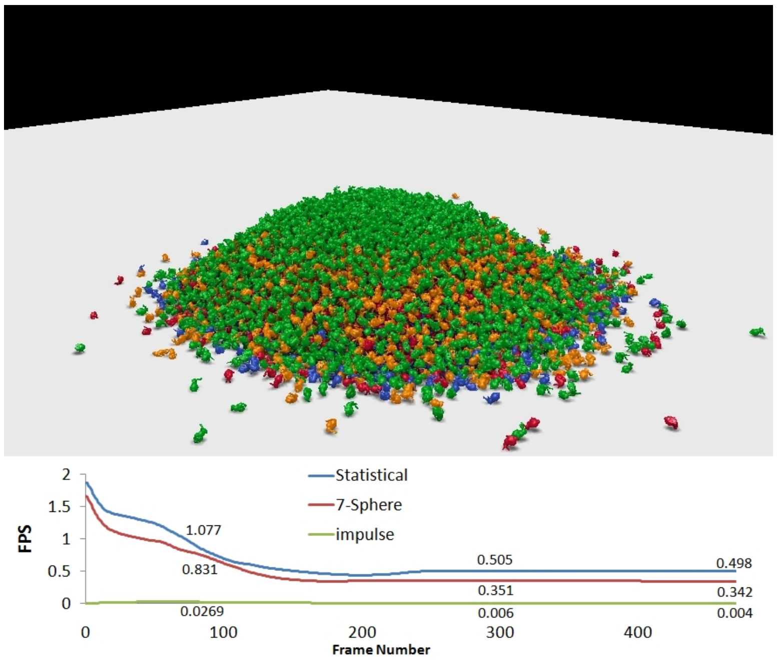


Figure 3.7: Statistical simulation of 10000 bunnies.

3.4.2.1 Performance of the Statistical system

In a variety of test situations, we achieve a clear improvement in speed with the simplified model versus all compared alternatives. To show the results, we provide a comparison of running times for a set of n randomly dropped bunnies in Table 3.1. Note that all methods implemented a full sweep-and-prune computation for the broad phase of collision detection.

For both the impulse and the statistical simulations, the vast majority of time was spent in the collision detection stage (this includes collision determination in the impulse system). The time spent on collision response, and in the statistical simulation on resting conditions, was negligible (typically orders of magnitude lower) than the time spent on collision detection.

As the table shows, compared to full impulse simulations of both the full model

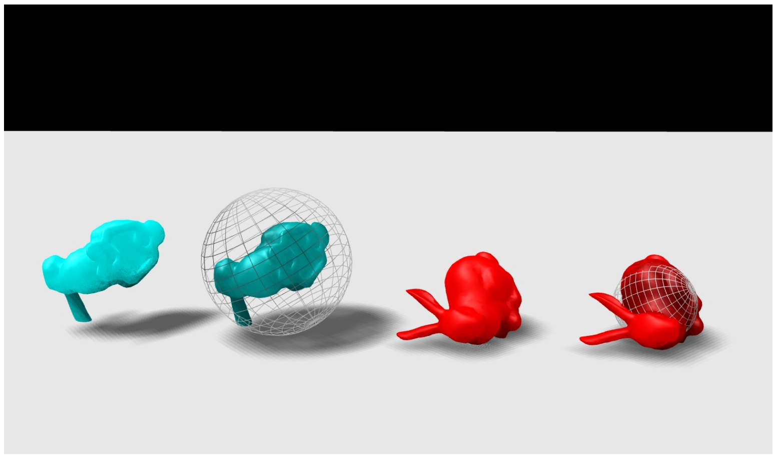


Figure 3.8: At left, the pink bunny model has come to rest without resting conditions applied, and at right the yellow bunny is shown with resting conditions applied. The same model is shown both with and without is colliding sphere. Note that the resting conditions result in the bunny having a more natural final orientation and collision sphere.

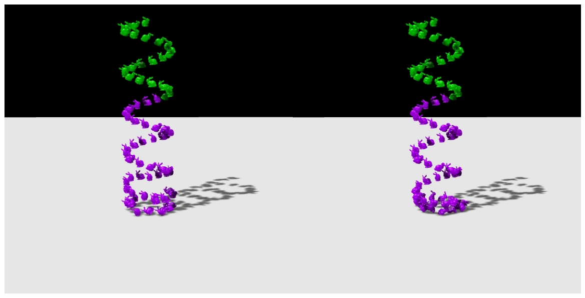
and a decimated model, our statistical simulation performs far faster. We also compare to a simplified model based on replacing the bunny model with a set of 7 spheres. While in the best case we would anticipate timings as much as 7 times faster (since we have only a single sphere check, instead of seven), the overhead of the broad phase of collision detection (shared by both approaches) reduces our benefit somewhat. However, for both small and large simulations, our method still performs noticeably faster than the 7-sphere models. Also, note that while the 7-sphere example usually produces reasonable collision responses, it does not necessarily achieve reasonable rest states, as in our method.

3.4.3 Discussion and Limitation

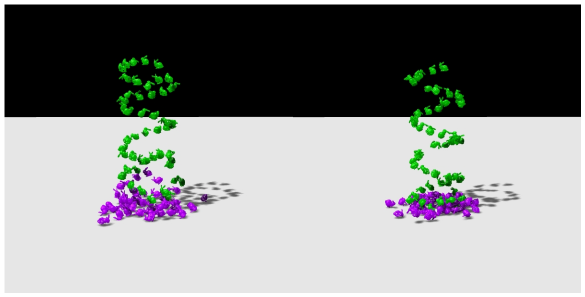
Our results show that the statistical system can provide major performance improvements. However, it is important to note that the statistical system carries significant tradeoffs. In particular, because the responses of objects obey only statistical response properties, close examination of collisions can result in observable inconsistencies (such as brief interpenetrations of objects or unrealistic response). Furthermore, collision responses that rely on objects being in very particular configurations (e.g. axis aligned starting positions) may not be faithfully captured. However, in exchange for this lack of fidelity we are able to produce responses that obey all statistical properties, and can be computed at much higher frame rates. For situations where a simulation forms the main focus of an animation, the method may not be appropriate, since small errors will be rapidly noticed. But, in situations where the simulation is either sufficiently obscured/unimportant/less important, etc., our method should work well.

The loss of accuracy is perhaps most evident in the handling of object-object collisions. Collecting data from object-object collision examples is much more challenging than doing it from the object-plane case, due to the vast number of possible configurations between the two objects. Our simplification — using the object-plane response — will lead to artifacts that may be quite noticeable in some situations. One such artifact is that after a collision response, the two objects will always move away from each other. Especially in large collections of objects, stacking may look unrealistic as a result of the object-object collision artifacts at the pairwise level.

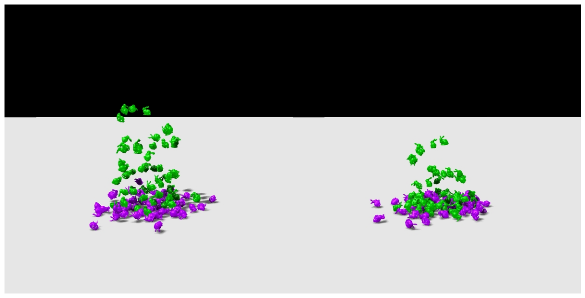
As the number of objects increases significantly, one would expect the broad phase of collision detection to dominate the narrow phase. While our statistical approach should still be faster than the alternatives, the speed benefit will be reduced somewhat, and thus the quality/speed ratio may shift in favor of alternative methods. Still, as we have shown in our timings; for examples that are feasible in real-time currently and in the near future, our approach should still provide a clear benefit over the alternatives.



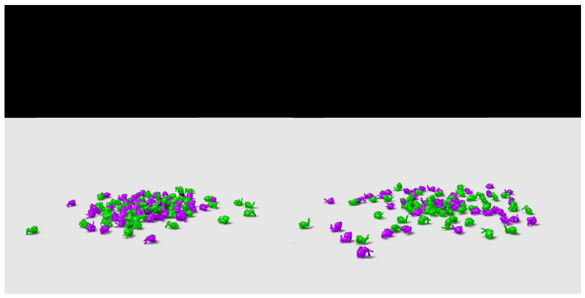
a



b



c



d

Figure 3.9: Simulation of 100 bunnies dropped to a plane. The left side of each image shows a statistical simulation, and the right side shows an impulse simulation. The statistical simulation is more than 50 times faster.

4. MIX STATISTICALLY-BASED AND PHYSICALLY-BASED SIMULATION

4.1 Introduction

In this chapter, we describe a method to combine our statistically-based rigid body simulator and a physically-based rigid body simulator to broaden its usability. Statistically-based simulation and physically-based simulation each have their own benefits and drawbacks. Physically-based simulation gives high-quality results, but at a slower speed, while statistical simulation does the reverse. In many situations, an all-or-nothing approach is inadvisable; we want additional speed, but are not willing to sacrifice quality everywhere in the simulation.

We try to solve this problem by using a mixed simulator which combines a statistically-based simulator with physically-based simulation (referred as an impulse-based simulator below). In the mixed simulator, according to an object's visual importance or different necessity in the scene, we can specify different simulation methods. Overall, the mixed simulator offers a non-uniform simplification, creating plausible rigid body simulations and achieving better performance as well.

4.2 Mix Statistically-based and Physically-based Simulation

4.2.1 Overview

To merge these two methods, we give every object an additional property, behavior preference, that is either physically-based or statistically-based. Given a pair of colliding objects, the preference of the objects is used to pick a proper collision handling approach. Algorithm 1 gives an overview of the collision handling algorithm. The first step is to perform a broad-phase collision check to find out possible colli-

Algorithm 1: Collision Handling

```
process broad-phase collision detection;
for each collision pair, $cp_i$ do
    update the behavior types of the objects in $cp_i$;
    if Dispatching($cp_i$) == Impulse then
        $Q_{impulse}$.add($cp_i$);
    else
        $Q_{statistical}$.add($cp_i$);
    end
end
process narrow-phase collision detection and response in $Q_{impulse}$ and
$Q_{statistical}$;
```

sion pairs for the next step. Then, according to the target an object might collide with, we update the behavior type of that object. The next step is to dispatch these pairs to different behavior queues. The final step is to process each collision behavior queue with the corresponding simulation method.

4.2.2 The Behavior Preference of Objects

In the mixed simulator, we have two categories of objects, "must-impulse" objects and "prefer-statistical" objects. A must-impulse object can only perform impulse-based behavior, which means it will use the full geometry model as the narrow-phase collision detection model and use an impulse-based approach to handle collision response. Alternatively, a prefer-statistical object might perform either statistical-based behavior or impulse-based behavior. These two kinds of objects both have an axis-aligned bounding box (AABB) for broad-phase collision detection and a full resolution collision model corresponding to their shapes. The prefer-statistical objects have an additional variable-radius sphere in the collision model, the same as in the sphere model we use in statistically-based rigid body simulation. The prefer-statistical object will choose a collision model (full resolution or sphere) depending

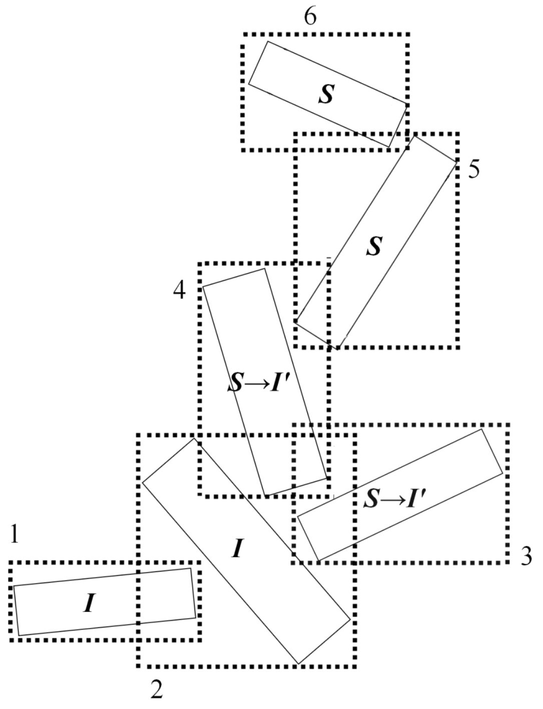


Figure 4.1: Collision pairs. For example, pair (1,2) is an (I,I) pair and pair(3,4) is an (I',I') pair.

on which behavior it follows.

We use three symbols to distinguish the cases above: I for a must-impulse object performing impulse-behavior, I' for a prefer-statistical object performing impulse-behavior, and S for a prefer-statistical object performing statistical-behavior. We can customize useful rules to decide when S objects will be updated to I' objects and when I' objects will come back to S.

Since we want I objects to always perform impulse-behavior, we have to update any S object to an I' object if the S object has a chance to collide with the I object. For this reason, once the AABB of a S or I' object touches the AABB of any I object, the S or I' object will be treated as an I' object; otherwise it will be a S object. Each time we have finished the broad phase collision check, we will apply this rule to update the status of each object.

We denote a broad-phase collision pair by the combination of the types in two objects. For example, an (S, I') collision pair means this collision is generated by a S object and an I' object. This combination will be used to select the corresponding collision handling approach in the narrow phase.

4.2.3 Dispatch Collision Pairs

All broad-phase collision pairs will be dispatched to different queues. A specific narrow-phase collision detection and response method will be assigned to each queue for testing and solving its own collision pairs. Figure 4.1 demonstrates all possible combinations of collision pairs, including (I, I), (I, I'), (I', S), (I', I'), and (S, S). There is no (I, S) pair because of the updating rule mentioned above. After updating the behavior type of objects, we will examine all collision pairs and dispatch them to different queues. For example, since we demand the pairs containing an I object must perform impulse-based collision handling, (I,I) and (I,I') will be inserted into a queue that will process by an impulse-based collision method, while others will be inserted into a different queue using statistical-based processing. Note that this is not the only way we can do dispatching, and it is very easy to choose different dispatching schemes, as we will see below. The only requirement is that the object has to provide all information the collision detection/response method needs. As an example, consider (I',I') interaction. Both I' objects have been created from S objects. We have the option of colliding these objects using either the full impulse method of the I representation, or the statistical method using the S representation (or even some other method). We can choose which queue to dispatch the pair to based on the importance assigned to those objects.

In addition, we can easily add more queues for dispatch. As a simple example, if we want to incorporate a particle-based approach to handle some unimportant pairs, we just dispatch those pairs to a new queue and use a particle-based approach to handle them.

4.3 Add Simulation Level of Detail to The Mixed System

SLOD is a good strategy to battle against the tremendous amounts of time that can be taken by massive simulations. In massive simulations, users will typically not notice fine details everywhere in the scene. Instead, a user might notice the atmosphere of the whole simulation first and then focus on some interesting points. Therefore, by regulating the amount of detail in the simulation, it is possible to capture the visual impression of a simulation for a user, as well as some important details, while achieving significant speedups.

Simulation level of detail (SLOD) can be achieved easily in our mixed rigid body simulation system. We adjust the level of detail of the scene primarily by varying the dispatch function used for colliding pairs. We describe here a basic three-level SLOD scheme that we found worked well in practice, though it is possible to create more levels of detail by providing more changes in the dispatching scheme.

In our SLOD scheme, each LOD has its own dispatch strategy, as shown in Table

Table 4.1: The dispatch strategies for three LODs in our system. We implement three different methods for solving the narrow-phase collision detection and response. By managing the dispatch strategies, we can adjust the quality over performance rates in different LODs.

Level	Impulse	Statistical	Particle
1 (closest)	(I,I) (I,I') (I',I')	(I',S) (S,S)	
2	(I,I) (I,I')	(I',I') (I',S) (S,S)	
3	(I,I) (I,I')		(I',I') (I',S) (S,S)

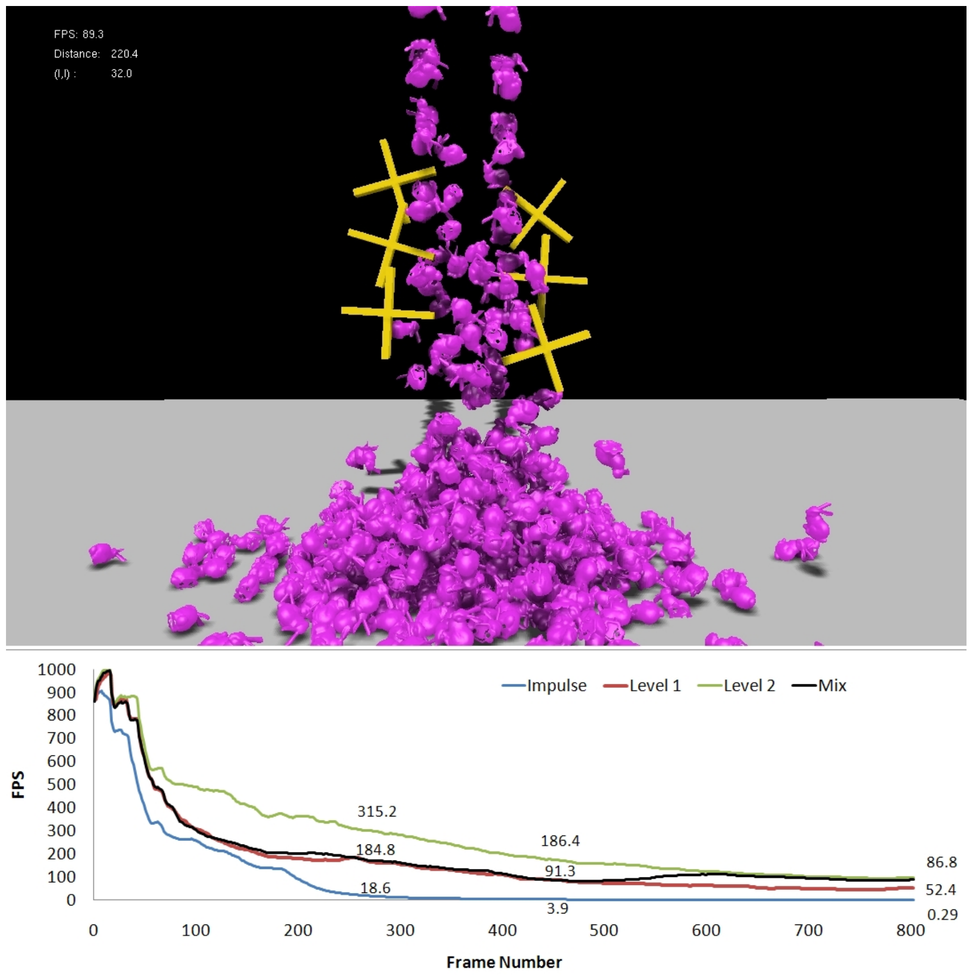


Figure 4.2: A simulation example of 500 falling bunnies pass through the cross objects and pile on a plane. We apply SLODs for this example. The blue, red, and green lines illustrate performance of pure impulse, pure LOD1, and pure LOD2, respectively. The black line shows performance of zoom out from LOD1 to LOD2.

4.1. For each level, our mixed simulator offers three dispatching options: impulse, statistical, and particle. Each option provides a different method for solving the narrow-phase collision detection and response for individual pairs. Notice that the LOD can vary from pair to pair; we do not use a uniform computation model to take care of a single level. The particular dispatch option is chosen by combining the preference (Must-Impulse or Prefer-Statistical) with selection rules that vary throughout the scene. The preference is a sort of "hard constraint" on the dispatch that can be used (limiting available options), while other selection rules form "soft constraints" that choose which of the available dispatch methods is most appropriate for that point in the scene. By considering an object's preference and level selection rules together we can regulate the detail of simulation in the scene more plausibly.

Assigning proper preference for objects is an important issue in our SLOD scheme, since this constrains the possible simplifications that are allowed. If we initialize every object as a must-impulse object, then the simulation will be the "exact" case, and we will have no speedup. On the other hand, if every object is a prefer-statistical object, the simulation will be biased to an extremely simplified case. In a typical scene, however, there are usually "key" parts of the scene that tend to draw the viewer's focus. Assigning these important objects or eye-catching objects as must-impulse objects and leaving other minor objects as prefer-statistical objects provides a good basis for SLOD. Figure 4.2 shows a result of our Level 2 SLOD strategy. The crosses are I objects while bunnies are S objects. Only the (I,I') pairs, or cross-bunny pairs, will be treated by a impulse approach.

Besides assigning preferences, a variety of selection rules can be used; we use rules analogous to other SLOD and geometric LOD methods. The distance between an object and the camera is our primary criterion for level selection. However, there are other options that could be incorporated as well, including an object's flying speed or an object's position in screen space.

For a particle-based simulation LOD, we let (I',I'), (I',S), and (S,S) be processed by a particle-based collision detection/response method. Although I' and S objects are prefer-statistical objects, we still can treat them as particles (spheres). The resting control will be eliminated. The collision response is changed from a table lookup approach to an even simpler particle collision response. Only the collision detection approach is the same as the statistically-based approach (compare the sum of radius and their distance).

Combining these methods gives us the ability to achieve significant speedups by replacing less important and less obvious portions of the scene by extremely simplified simulation (i.e. statistical simulation and particle simulation). This also provides an SLOD framework that is very flexible, accommodating a variety of dispatch strategies and collision methods.

4.4 Results

In this section, we show the implementations and results of our mixed simulator.

4.4.1 Implementation

In our mixed simulator (referred as the *mixed* system), we combine the impulse system and statistical system. For collision detection, we use AABB-based sweep-and-prune in broad phase. In the narrow phase, we could use SWIFT++ for impulse cases or sphere-based collision for statistical cases. Note that SWIFT++ combines both broad and narrow phases, and we had to work around their code to use our own broad phase computation in the mixed simulation; as a result, the combination of our broad phase with Swift++'s narrow phase is less efficient than using Swift++ alone, but the efficiency loss is minor and does not affect any conclusions.

4.4.2 Results: The Mixed System

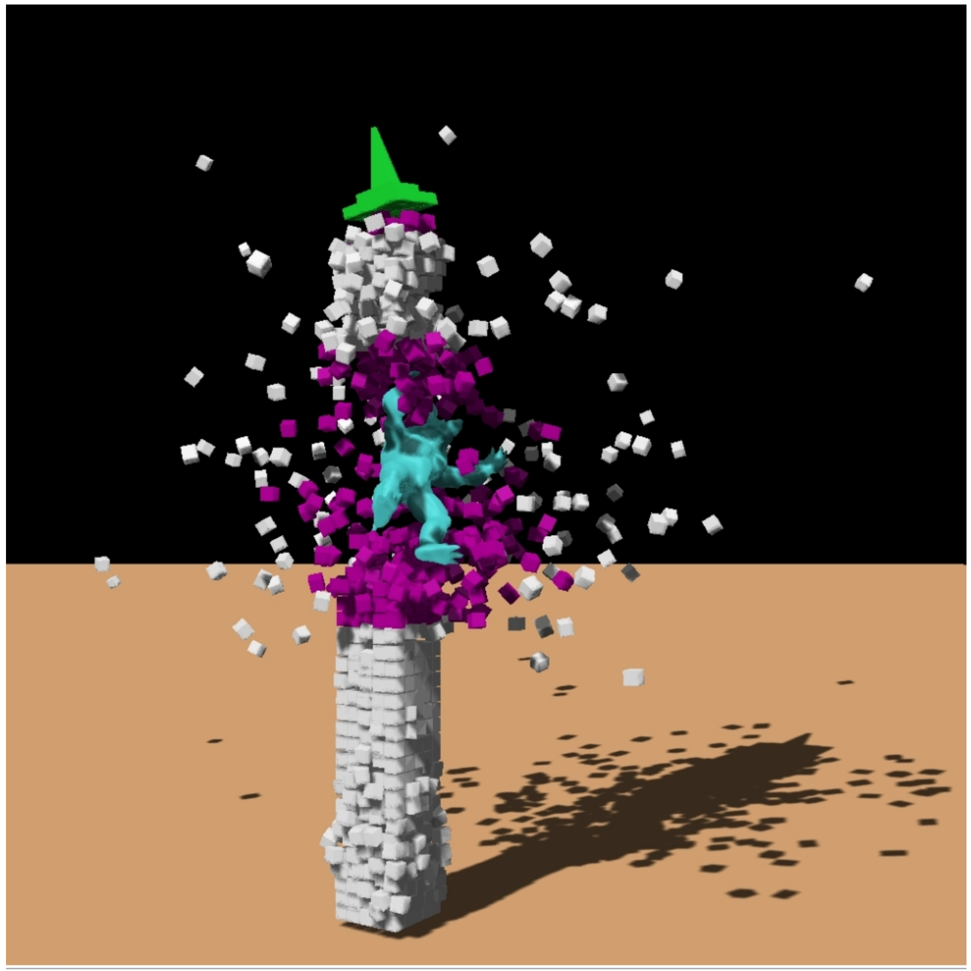
Results of the mixed system are presented in Figures 4.2, 4.3, 4.4, and 4.5. Relative timing comparisons can vary throughout the simulation, so we show performance graphs with each picture; statements about relative performance do not always apply to the entire length of simulation.

We illustrate our different dispatching methods with the example in Figure 4.2. We drop 500 bunnies from the air onto a plane. There are 6 cross-shaped objects, which are allowed to rotate along one axis only. The falling bunnies can collide with these cross shape objects, making them spin. Each bunny and cross has 4968 and 44 triangles, respectively. Generally, the focus of the scene is on the bunny-cross interaction; interactions between bunnies, and of the bunnies with the plane are relatively minor visual features. As a result, we set the six spinning crosses as "must-impulse" objects while bunnies are "prefer-statistical" objects. The collision pairs of a spinning cross and a bunny will be (I,I') and pairs of two bunnies will be (I',I'), (I',S) or (S,S). We thus achieve highly accurate simulation in the key area (where the bunnies are interacting with the crosses) while saving significant time by reducing computation in other areas. We test three dispatch strategies separately in this scenario: pure impulse (all objects, including bunnies, are treated as must-impulse), Level 1 LOD, Level 2 LOD, and switching from Level 1 LOD to Level 2 LOD (where the objects are dispatched according to the mechanisms of Level 1 or Level 2 of Table 4.1). In the Level 1 and Level 2 LOD tests, LODs do not vary at all in the simulation. In the switching case, we first start from a close-up view where the Level 1 LOD is in effect, and then slowly zoom out to where only Level 2 LOD applies. Figure 4.2 shows performance comparisons for this example. As can be seen, using LOD methods provides significant speedup (more than an order of magnitude).

The scenario in Figure 4.3 is of an armadilloman hitting a building composed of 1000 cubes. We set the armadilloman, the top of the building, and the ground as "must-impulse" objects and all cubes as "perfer-statistical" objects. We test a customized dispatch method in this example. Only cube-cube and cube-ground pairs are handled by the statistical simulation, while all other pairs are handled by the impulse method. We capture most of the major visual effects, such the crashing of the building, the piled cubes as debris, and the interaction between the top, the armadilloman, and the debris. We maintain a rate of more than 25 fps throughout this example.

Figure 4.5 demonstrates that the mixed system is able to handle a massive simulation while maintaining fps at an interactive level (5 fps). 12000 cubes are composed as different structures at the beginning. Then, we use armadillomen to hit them. The LOD3 dispatch strategy is used in this example so that all cube-cube collision pairs are handled by our narrow phase particle-based collision detection and response method.

The scenario in Figure 4.4 mimics an earlier rigid body simulation demonstration ([24]) by dropping 1000 rings onto 25 poles. We examine several different scenarios here, to demonstrate the flexibility of the mixed system. At two extremes, we examine cases of all impulse collisions (runs very slow) or all-statistical simulations (has obviously incorrect results). For the mixed cases, the poles are "must impulse" objects while rings are "prefer-statistical" objects. Again, because the pole-ring interactions are the most visually critical, we perform those computations precisely; several examples where rings have gone around poles are readily visible. Less important parts of the scene, such as the details of rings within the piles are performed with less accuracy. We then examine two sub-cases: one where the ground plane



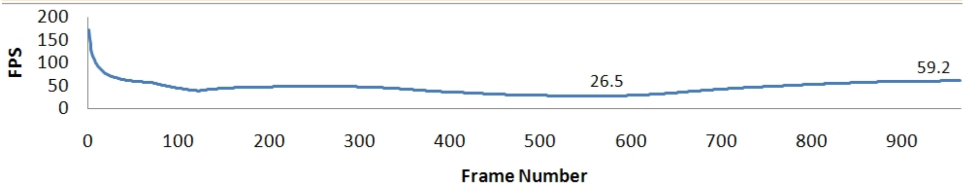


Figure 4.3: An armdailloman hits a building, which is composed of 1000 cubes.

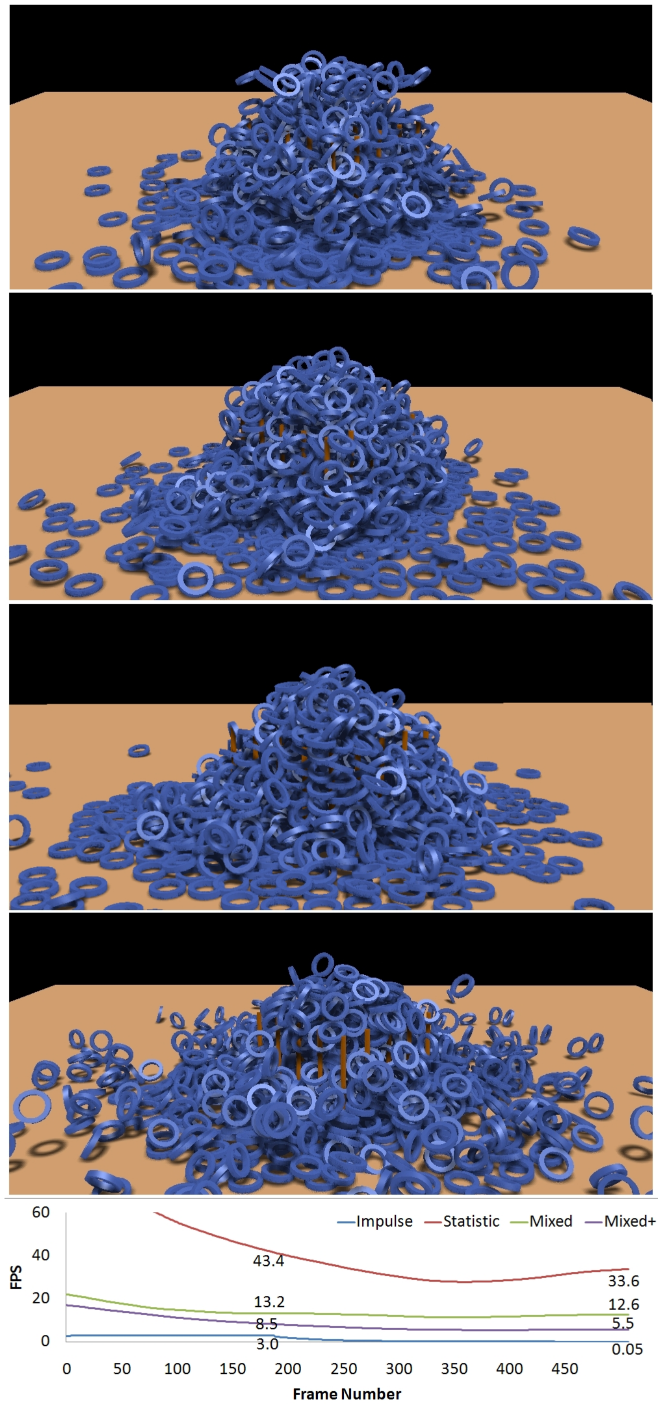
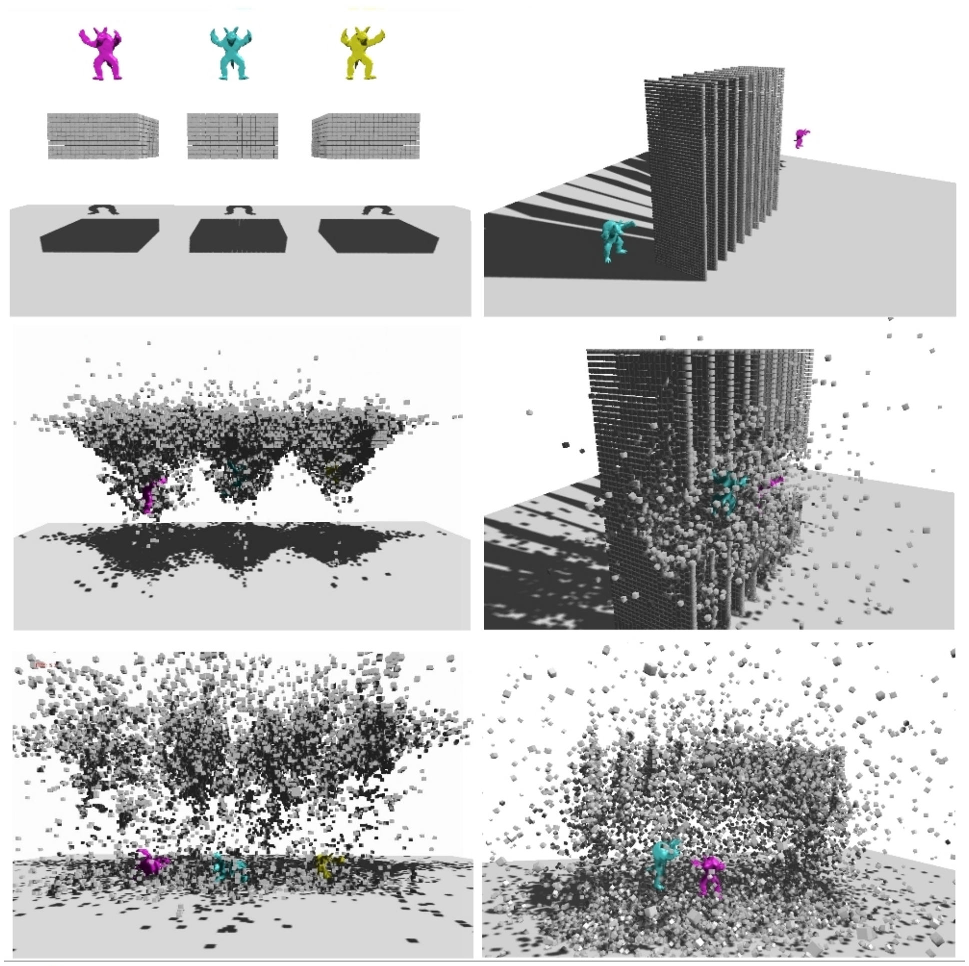


Figure 4.4: Simulation of 1000 rings dropped onto 25 poles. The top image (impulse) shows an intermediate frame from an impulse-only simulation, which runs very slowly. The second image (statistic) is a frame from a statistical-only simulation (which looks very incorrect, especially animated). In the third image (mixed), poles are must-impulse objects but rings and the ground are prefer-statistical objects. The fourth image (mixed+) is a frame from a mixed simulation where the ground is also must-impulse.



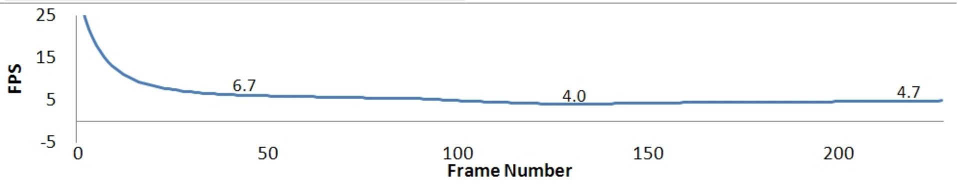


Figure 4.5: Massive rigid body simulations of 12000 cubes hit by armdaillomen. The performance chart shows we can reach interactive rates (5 fps) in the mixed system.

is “prefer statistical” and one where it is “must impulse.” Both methods are significantly faster (sometimes exceeding two orders of magnitude) than the “correct” impulse-based simulation of this scene. The more accurate ground plane cuts the frame rate by a factor of three; this allows one to choose the level of accuracy based on the time available and the importance of that interaction.

Table 4.2: Videos in our user study. The table shows the methods and geometric models in the testing scenarios.

Scenario S_1 (1000 bunnies)

	Simulation method, Collision models	label
$video_1$	Impulse method, Fine models	$V_{S_1,impulse}$
$video_2$	Impulse method, Approx. the bunny by 7 spheres	$V_{S_1,simplified}$
$video_3$	Statistical method, 1 sphere	$V_{S_1,statistical}$

Scenario S_2 (Rings and Rods)

	Simulation method, Collision models	label
$video_1$	Impulse method, Fine models	$V_{S_2,impulse}$
$video_2$	Impulse method, Approx. the ring by 1 sphere	$V_{S_2,simplified}$
$video_3$	Mixed method, Approx. the ring by 1 sphere	$V_{S_2,mixed}$

4.4.3 User Study

We have conducted a user study to evaluate the effectiveness of our proposed approach. The user study was mainly conducted by Dunghui Han. The goal of this user study is not only to evaluate the quality of our results but also to discover viewing patterns and other characteristics when people are watching computer-generated simulations. Due to the broad goal of this user study, we focus the discussion on the results related to our simulators (including the statistically-based and the mixed simulator) instead of describing the whole user study in this dissertation.

4.4.3.1 Videos

We used two scenarios to compare the quality of our results, one for evaluating our statistically-based simulator and the other for our mixed simulator. The setting of the first scenario(denoted as S_1 later) is similar to the one shown in Figure 3.7, but there are only 1000 objects. For evaluating our mixed simulator, we choose the second scenario from the example shown in Figure 4.4 (denoted as S_2).

For each scenario, we prepared three videos that were generated by three different simulation approaches: a physically-based method with fine geometric models, a physically-based method with simplified geometric models, and our method (the statistical or mixed method). The first simulation approach represents a traditional physically-based approach, which generates good results but is very slow. The second simulation approach uses the same simulation method as the first one but improves the performance by using simplified geometric models. And the third one is our proposed method. Table 4.2 is the video list. Note that there are more than six videos in the whole user study, but the rest of them are designed for evaluating or discovering other phenomena so we focus our discussion on these six videos.

4.4.3.2 Settings

We let subjects watch videos and rate the videos. Meanwhile, an eye tracker recorded the eye movements of each user. We showed each subject all video clips twice with random sequence. After viewing a video, a subject was asked to rate that video, from 1 to 9. The score indicates the realism of the physical behavior in the video; the higher, the better. There were 32 subjects participating in this user study.

4.4.3.3 User Study Results

According to the analysis results, we have not seen a significant score difference between the videos made by our methods and the videos made by the impulse-based method with fine models. Based on the results, we believe that our methods are able to produce large scale rigid body simulations plausibly.

We used two-way Analysis of Variance (ANOVA) to determine whether the simulation methods affect the scores. The treatment in the ANOVA analysis is the simulation method. For each scenario, we compared our method ($video_3$) to the normal impulse method ($video_1$). We also compared the simplified impulse method ($video_2$) to the normal impulse method ($video_1$). Table 4.3 shows the mean score of each video. Table 4.4 is the ANOVA results. For each analysis case, we considered that there is a significant score difference for the video pair if the p-value is lower than the significance threshold. Significance threshold was set to 0.05 (the alpha value). If there is no significant score difference for the video pair, we believe the quality of both videos are in the same plausible level.

In scenario S_1, both videos generated by our method and the simplified method are as plausible as the video generated by the impulse-based method. In scenario S_2, the video generated by our mixed method generated is plausible, while the simplified method is not. Since the scene in S_2 is more complex than the scene in S_1, our mixed method has a better ability to handle it than the the simplified impulse method.

Table 4.3: The averaged scores of the videos in our user study

Scenario S_1 (1000 bunnies)		Scenario S_2 (Rings and Rods)	
$V_{S_1,impulse}$	5.81	$V_{S_2,impulse}$	5.69
$V_{S_1,simplified}$	5.81	$V_{S_2,simplified}$	4.53
$V_{S_1,statistical}$	5.44	$V_{S_2,mixed}$	5.48

In scenarios S_1, we noticed that the simplified impulse method gets a better mean score than our statistical method. Due to the camera setting of that scenario, we expect some users might have noticed some errors from our simplified method for object-object collisions and our method for resting control so the overall score might go down a little bit.

4.4.4 Discussion

Our results show that the mixed system broadens the usability of our statistical simulation method. The goal of the mixed system is not to produce an absolutely correct simulation result, but rather one in which the overall simulation behaves in a plausible way. We do achieve this by maintaining accuracy at the "important" points of the simulation, while allowing less accurate but statistically correct results in other parts of the scene.

Because of the possible errors introduced by the statistical method, the mixed system would still be inappropriate in a setting where high accuracy is desired across the entire scene. We see two main application areas for the mixed system. First, since we typically achieve more than one order of magnitude (and often more than two orders of magnitude) improvement in performance, the mixed system may be appropriate for massive simulations where a full accurate simulation is just not oth-

Table 4.4: The ANOVA results of the user study. We consider that there is a significant score difference for the video pair if the p-value is lower than the significance threshold(0.05)

video pair	F test	p-value
$(V_{S_1,impulse}, V_{S_1,simplified})$	$F_{1,31} = 0$	1.000
$(V_{S_1,impulse}, V_{S_1,statistical})$	$F_{1,31} = 3.207$	0.083
$(V_{S_2,impulse}, V_{S_2,simplified})$	$F_{1,31} = 20.028$	< 0.0001
$(V_{S_2,impulse}, V_{S_2,mixed})$	$F_{1,31} = 3.166$	0.085

erwise feasible. Second, in many interactive situations, there is a need for speed, as well as a limit on the level of fidelity required across an entire scene; it is likely that only a portion of the scene is important to simulate accurately. For such interactive scenes, the proposed method should work well.

It is important to note that our mixed system allows significant variability in the tradeoffs allowed. By varying which portions of the scene are “must impulse” vs. “prefer statistical”, we can achieve various performance/speed tradeoffs (the ring example demonstrates this). Likewise, it should be noted that different dispatch strategies can be adapted to particular scenarios. The building example demonstrates that for a constrained type of example, we can develop specialized dispatch strategies that allow a quality/performance tradeoff for a particular scenario.

5. MODELING PILES OF OBJECTS VIA ANGLES OF REPOSE*

5.1 Introduction

In this chapter, we introduce a method for representing piles of objects in large multibody simulations. Object piling and stacking presents one of the more interesting phenomena, but also one of the more computationally intensive situations, in such large simulations. For this reason, a method that can handle piling conditions at faster rates with more user control would be desirable. This chapter aims to provide such a method.

There are two major reasons for implementing a specific piling approach. First, by representing piles of objects, we can deactivate objects within the pile, saving significant time. This offers improvement over standard methods for deactivating objects, based strictly on object momentum.

Second, and perhaps more important, stacking behavior that one may wish to see is often influenced by friction among the simulated objects. Simulating static and dynamic friction accurately is a much more complex process than simple impulse-based collision response, and is thus avoided in many simulations. However, by avoiding friction, one is limited in the types of stacking and the shapes of piles that can be achieved. Our piling method allows us to specify the shapes of piles that can be obtained, allowing for a more realistic or more art-directed look to emerge.

The main result presented in this chapter is a model for piles of objects in which layers of objects are explicitly represented. We present a method for determining where these layers should be and updating the layers as objects pile up. The only change to the fundamental simulation is a new criterion, based on the pile structure,

*Part of the data reported in this section is reprinted with permission from Piles of Objects by Shu-Wei Hsu and John Keyser, 2010. ACM Transactions on Graphics, Vol. 29, No. 6, Article 155, Copyright 2010 by ACM.

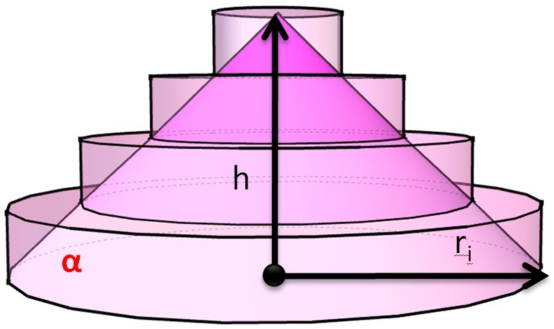


Figure 5.1: An example of our stacking structure.

for marking objects “asleep” and stopping their simulation. This not only enables us to improve computation time, but also lets us provide greater user control in the structure of the piling shape.

5.2 Generating Piles

To generate piles, we create a spatial structure that we refer to as the “stacking structure.” This structure will be used to explicitly represent the pile(s) of objects. Objects that are determined to be part of the pile are put to “sleep,” i.e. they are no longer simulated, unless later reactivated.

5.2.1 Stacking Structure

Our stacking structure is a spatial data structure that is used to track the density with which objects are packed in a pile. Figure 5.1 shows an example of the structure. A cone shape is used to track the stacking activity. The stacking structure location may be specified manually (when a particular pile location is known or desired), or set based on where small sets of objects initially land. This cone shape is defined by an angle of repose, α, that is the one user-defined parameter. α can be determined by measuring the shape of a pile of objects in an precomputed simulation, or more typically may be set by the user to a desired value. In this way, α can reflect the contributions of frictional components that are not directly simulated.

Consider a simulation of N objects of a single type. The volume of each object is v_{obj}, where we use a conservative bounding volume to approximate volume. A conservative measure for v_{obj} is preferable to an exact volume, since the volume measure should reflect the amount of space the object occupies in a pile, rather than the true object volume. We typically use a bounding box for this estimate, but observe no significant difference in behavior of the simulation when similar estimates are used. We also determine the maximum length of the object in any direction, D, which we can approximate by a bounding box diagonal. We assume those objects will pack together to form the cone, making the volume of the cone approximately Nv_{obj}. Given an angle α for the cone, we have a relation between the base radius r and height h such that $\tan\alpha = h/r$. We can thus determine the base radius from:

$$Nv_{obj} = \frac{\Pi}{3} r^3 \tan\alpha \tag{5.1}$$

We generate a spatial representation of this structure bottom-up. We cut the cone into k layers, where $k = \lfloor h/D \rfloor$, ensuring that no object can span more than two layers. For each layer i there is a corresponding radius r_i and volume v_i, obtained by simple geometry. The maximum capacity c_i of layer i is set to v_i/v_{obj}. Thus, a stacking structure is defined by (α, k, $r_1 \ldots r_k$, $c_1 \ldots c_k$), although many of these values can be computed, rather than stored.

5.2.2 Growing the Pile

During simulation, more and more objects will fall into the stacking structure, causing it to grow. We use an occupation ratio to determine whether the structure needs an update or not. Let m_i be the number of objects in layer i of this structure.

To collect m_i, we check the location of the center of mass of each object. We also compute the density of the layer, $\rho_i = m_i/c_i$. These values (m_i and ρ_i) are maintained continuously throughout the simulation.

The occupation ratio ρ can be computed as $\sum(m_i)/\sum(c_i)$. If $\rho > 0.5$, we will update the structure. To update the structure, we simply regenerate the structure again with $N = 2\sum(c_i)$ (i.e. approximately doubling N); the angle is still α.

The center of a pile can be updated over time by gradually shifting the x, y coordinates of its center to reflect the average of those objects in the pile.

5.2.3 Causing Objects to Sleep

As in other simulation approaches, we achieve efficiency by deactivating objects that are either hidden from view or not moving. We refer to this as putting the objects to "sleep." Determining whether an object should go to sleep relies on three things: how dense the layer the object belongs to is, how close the object is to the center of the layer, and how slowly the object is moving. If an object is in a dense layer, close to the center, and slow enough, we can aggressively force it to go to sleep.

Assume an object is located on layer i. We use $\rho_i = m_i/c_i$ as the first measurement, which is the density of layer i.

To measure closeness to the center, we use the following equation:

$$d = \max(r_i - \sqrt{p.x * p.x + p.y * p.y}, 0)/r_i \tag{5.2}$$

where p is the position of the object's center of mass, with z being vertical (we assume the pile is transformed to lie at the origin). If the object is located on the center of the layer, the value will be one; if the object is not inside the cone, the value will be zero.

The last measurement, speed, is computed as $\upsilon = 1/(1 + |v|)$, where $|v|$ is the magnitude of the object's linear velocity. υ ranges from 0 (at infinitely high velocity) to 1.

To determine when an object goes to sleep, we multiply these three measurements together (i.e. $\rho_i d\upsilon$) to get a sleep score. If the score is greater than some threshold, τ, the object is ready to be moved to a sleep state. This score will tend to make objects go to sleep from the center of the pile toward the outside. However, we also want to ensure that objects sleep bottom-up (so that objects on top of the pile can move if those underneath are still moving). To do so, we check whether all levels $k < i$ contain sleeping objects (i.e. are dense and contain objects close to the center). If so, we allow the object to be put to sleep.

5.2.4 Avalanching and Waking

The procedure outlined above is sufficient to allow objects to pile up, and stack nicely in ideal situations. However, there is nothing in the above that would prevent a small pile to form, objects to be put to sleep, and then more objects pile on top, creating unbalanced and unrealistic piles. There needs to be a way to "wake" objects, as well as to detect conditions in which the pile shape is not being maintained. We use an avalanching technique to detect such cases.

Avalanches form when the upper levels of a pile cannot be supported by those underneath. This will happen when a pile temporarily exceeds its natural angle of repose. Note that this can occur at any point along the pile; it is a local effect on the pile, not a global one. In our pile structure, this is detected by examining the density of each layer of the pile. We trigger an avalanche whenever we detect that a given layer has density greater than some threshold (we use 0.8 in our implementations). When an avalanche is triggered, we temporarily wake up all objects, and allow them to undergo motion. When objects are awoken, they will simulate for several timesteps before they are checked against the sleeping threshold. This allows them to sufficiently resolve any forces or imbalances. Most objects not in the unbalanced region of the avalanche will quickly return to a sleeping state, but those in the unbalanced area will fall to lower areas. The result is an effect that one expects to see, where piles that become unbalanced have regions that slide down to more stable configurations.

Objects in the pile can also be woken when an external force (such as from a high-momentum colliding object) hits the pile. Such forces may propagate throughout the entire pile in a single time step, and thus it is necessary to wake the entire pile when such an event occurs. We use a momentum heuristic for waking in such cases—i.e. if an object with sufficiently high momentum hits a pile, the pile is woken so that the force propagation can be resolved.

5.2.5 Pile Boundaries and Overlapping

Pile growth can be constrained in various ways. If a scene contains fixed boundaries, the pile growth can be constrained so that no layers may extend past the boundaries (see Example 6 in Section 5.3.1). If the lowest layer reaches full density, but is touching the boundary, it is not expanded further. Also, a constrained pile's center can be moved vertically. This allows modeling of piles of objects within containers, where the pile will build in the container, fill it up (creating a rough cone over the container), and eventually spill over the sides.

Multiple piles can easily be used within a single simulation (see Examples 3 and 5 in Section 5.3.1). Note that single objects might contribute to the density measurement of more than one pile. Also, two piles whose centers lie near enough to each other and are of the same type can be combined into a single pile.

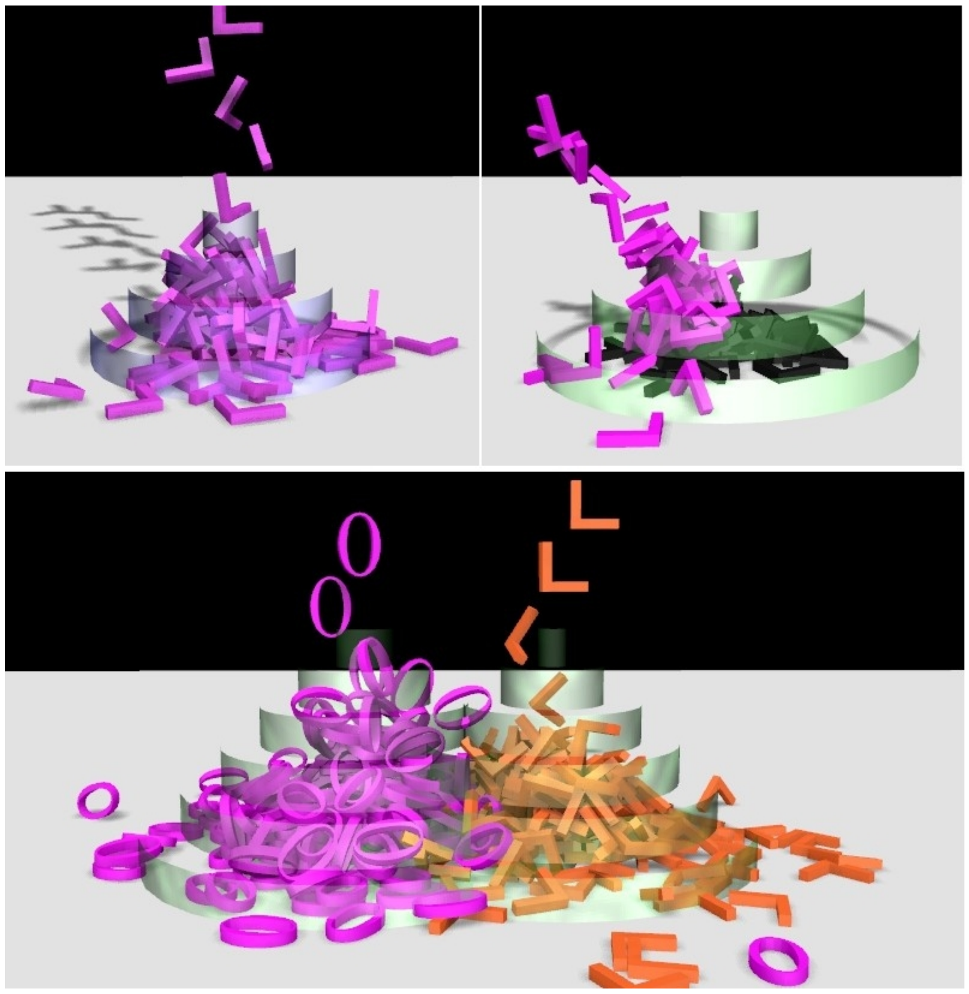


Figure 5.2: a. Upper left: Example 1—Drop in a Pile. b. Upper right: Example 2—Varying Drops. c. Bottom: Example 3—Multiple Piles. The semi-transparent cylinders show the layers of the stacking structure

5.3 Implementation and Results

We have implemented our piling approach within a rigid body simulation. We use Swift++ [18] for collision detection, and a basic impulse-based collision response. All results are collected on a system with a Core 2 Duo 2.66 processor, 4 GB of RAM, and an NVidia 9800GT card.

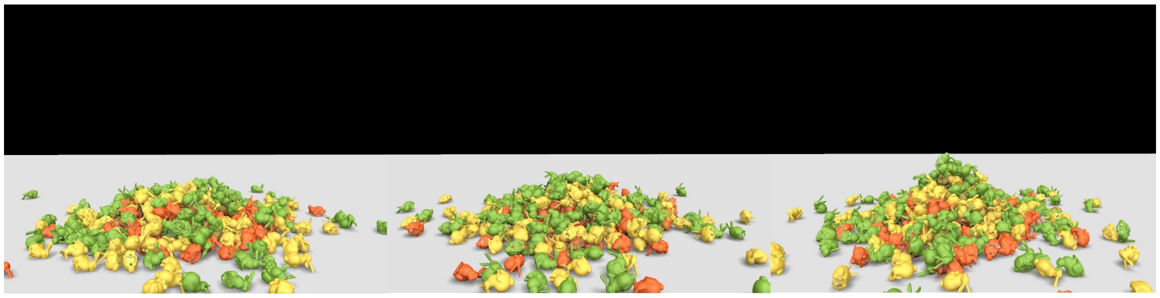


Figure 5.3: Example 4—Bunny Matrices. The results from three different runs, with three different angles of repose, are shown.

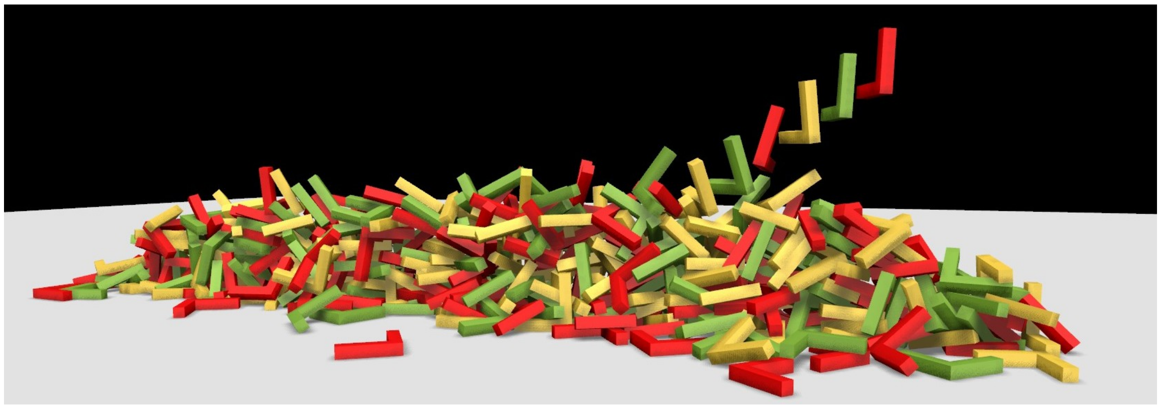


Figure 5.4: Example 5—L-shaped objects piling up.

5.3.1 Examples

To explore the usability of our piling approach, we use several different examples. Examples 1 through 4 use piles located where the first object hits, while Examples 5 and 6 use manually specified piles. We test the abilities of our method for handling large numbers of objects, multiple sized objects, and interactions with other objects in Examples 7, 8 and 9. Table 5.1 gives details of the settings and results of the examples.

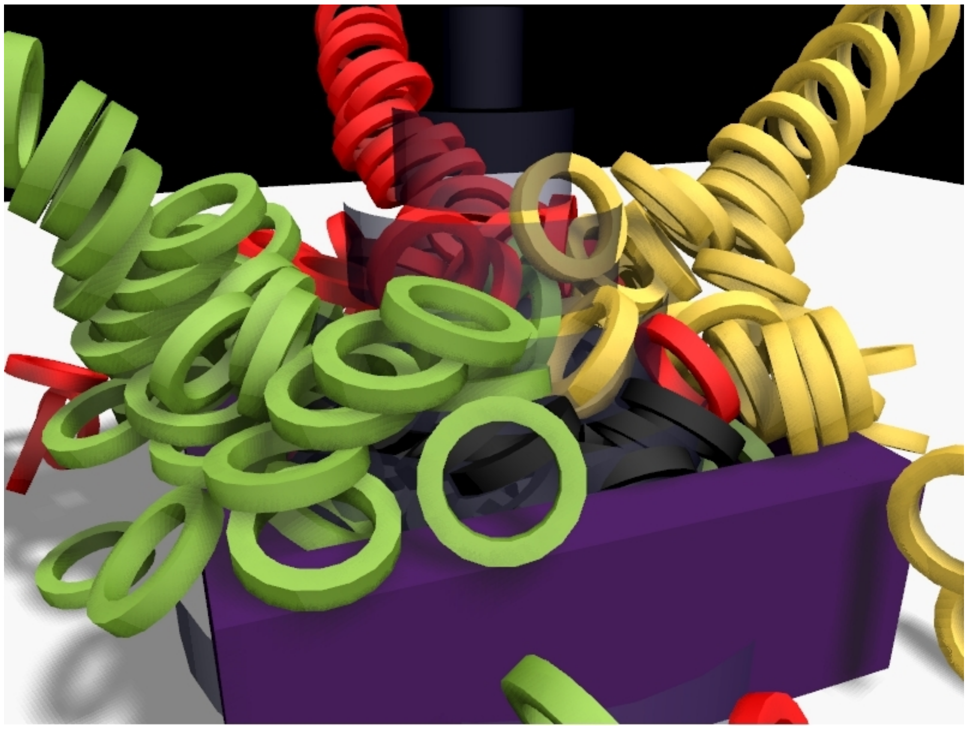


Figure 5.5: Example 6—Rings in a Box.

Example 1, Drop in a Pile (see Figure 5.2a): In this example, we drop several L-shaped objects onto a plane. Objects are dropped one-by-one directly down. This example demonstrates how the structure grows, how the objects stop, and how avalanches happen. In the pictures, we use transparent cylinders to show the layers of our pile structure. Sleeping objects are colored black. The result shows that sleeping objects generally build inside-out and bottom-up. Also, when the pile becomes too high, all objects wake up and the simulation undergoes an avalanche effect.

Example 2, Varying Drops (see Figure 5.2b): In this example, we drop objects from different directions, and vary the timing over which they drop. We start by firing one string of objects from a non-perpendicular angle to a plane. When the objects have piled up, we fire another string of objects from a different angle to that pile again. The result shows that the structure can easily deal with discontinuous

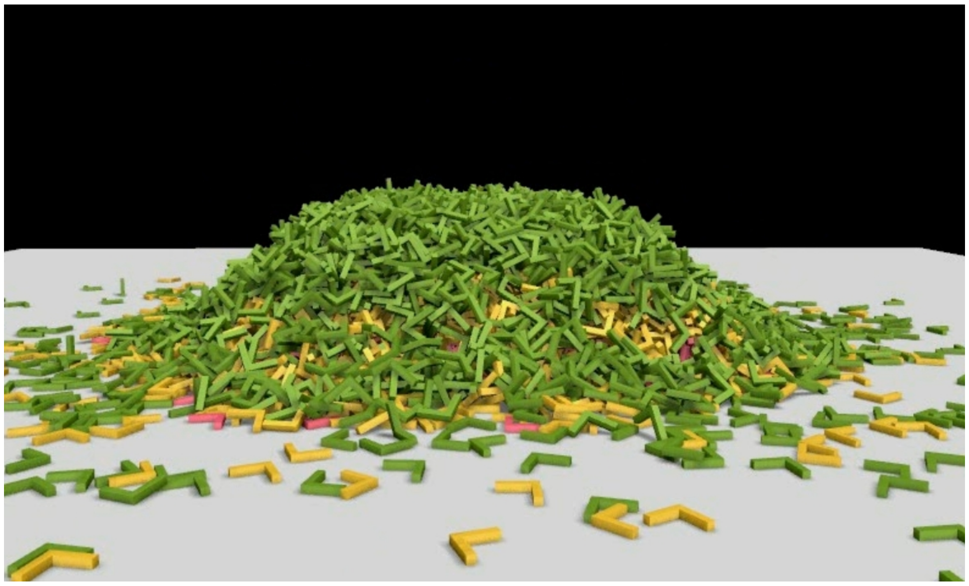


Figure 5.6: Example 7—10000 L-shape objects.

dropping.

Example 3, Multiple Piles (see Figure 5.2c): Another characteristic of our stacking structure is that it is able to use multiple stacking structures and to mix them together. In this example, L-Shape and O-Shape objects are used. Again, they are dropped straight, parallel, and close to a plane. We generate two stacking structures. The result shows that although objects in the two piles are put to sleep separately, mixing them together still creates a reasonable sleeping distribution, and piling behaves correctly.

Example 4, Bunny Matrices (see Figure 5.3): We drop a set of bunnies organized in a 3D matrix, in order to quickly form a pile. We use this example to demonstrate varying angle of repose in the piles. We run the simulation three times, using three different angles of repose. The smallest angle matches the angle of repose calculated from a prior run of the full simulation, and the other two use larger angles. The difference in angle is clearly seen, and all piles appear plausible. Our structure

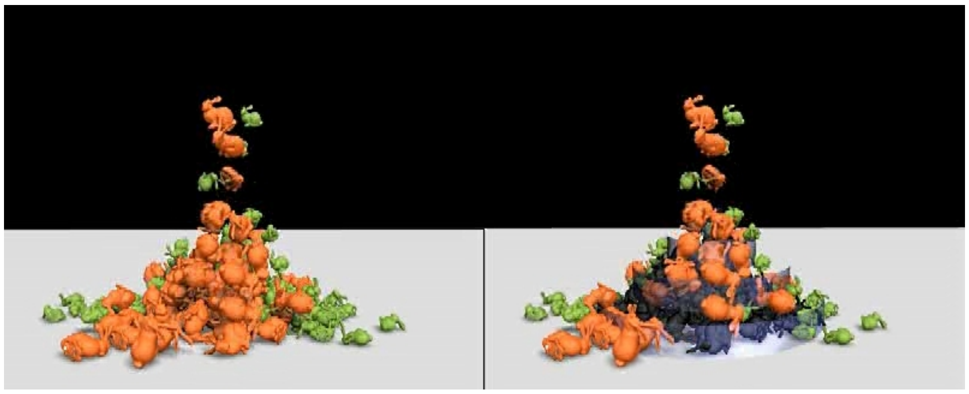


Figure 5.7: Example 8—Multiple sized objects.

handles this situation, however the performance improvement is limited, compared to other piling examples. The density of the top layer is always high when these matrices are falling down, therefore the structure cannot allow objects to sleep while objects remain at or above the top layer.

Example 5, Drop in a Line (see Figure 5.4): We drop a set of L-shaped blocks, where the drop point gradually moves back and forth along a line. Seven pile structures are used, forming a line. We use this example to demonstrate that piling can be used to create non-circular piles. That is, even though the individual piles might be generated from cylindrical stacks, the combination of piles does not have to yield something like a single pile.

Example 6, Rings in a Box (see Figure 5.5): Dropping objects into a container is one example of a possible situation for our simplified stacking method. We prepare a box and set a stacking structure in the center of this box. As objects pile up, the objects on the lower layers become occluded, making it safe to put them to sleep. In addition, we want objects to sleep only inside the box. To do so, once the size of the lowest layer exceeds the boundary of the box, the structure will stop growing.

Example 7 (see Figure 5.6): This example includes a larger pile structure ($10,000$ objects). It shows our method can handle large numbers of object.

Example 8 (see Figure 5.7): We use multiple sized objects in this example.

Example 9 (see Figure 5.8): An example of a pile being broken up after being hit by a high momentum object.

5.3.2 Performance and Error

The overall performance of the example scenes is summarized in Table 5.1. In addition, we show charts with the timing and error of Examples 5 and 6 in Figures 5.9 and 5.10, respectively.

5.3.2.1 Behavior

Visual examination of the various examples verifies that we are able to achieve the primary behavior we desired, namely the forming of piles of objects. Figure 5.3 illustrates different pile angles achieved from the same basic simulation. The simple impulse-based simulations that we use cannot, on their own, achieve the higher angle of repose that might often be desired, without resorting to much more expensive calculations of friction. Comparison to the full simulation without piling demonstrates that we are able to achieve desired stacking behaviors (i.e. taller stacks) that are not achieved by the basic simulation alone.

5.3.2.2 Timing

As can be seen in Table 5.1, there is a speedup from using the piling method, in all simulations. Figures 5.9 and 5.10 illustrate that this timing difference is not uniform over the course of the simulation. As one would expect, the performance improvement occurs when the objects start to pile up. When piles consistently grow (as in Figure 5.9), the performance improvement continues to grow. When a pile

	Example	Angle	# Frames	# Objects	Full Sim. (sec)	Piling (sec)	Speed-Up	Error (%)
1	Drop in a Pile	45	1818	400	4125.4	681.5	6.0x	5.8
2	Varying Drops	30	856	140	495.2	231.9	2.1x	20.8
3	Multiple Piles	45	2203	400	5571.2	1488.2	3.7x	5.1
4a	Bunny Matrix I	20	312	300	596.1	502.3	1.2x	9.3
4b	Bunny Matrix II	35	312	300	596.1	359.4	1.7x	16.0
4c	Bunny Matrix III	45	312	300	596.1	333.4	1.8x	4.2
5	Drop in a Line	45	928	400	1461.6	600.8	2.4x	9.6
6	Rings in a Box	60	1061	450	4904.2	2557.2	1.9x	9.9

Table 5.1: Overall timing results for examples. Angle is the angle of repose used in the example. # Frames and # Objects are the number of frames for the full simulation, and the number of objects used in it, respectively. The two timings are for the "full" simulation (without piling) vs. the simulation with piling, and Speed-Up gives the relative difference over the course of the entire simulation. The error computes the average difference in movement relative to the full simulation (relative error discussed in Section 5.3.2.4), over the course of the entire simulation.

reaches a maximum size (as in Figure 5.10), the performance improvement tends to level off, though it remains significant.

Comparing timings of the Bunny Matrix examples, we notice that the more piling (i.e. the greater the angle of repose), the greater the speedup. This is to be expected, since objects that are in the pile are deactivated more readily.

Our timing comparison is somewhat unfair, in that the full simulation does not deactivate any of the objects, even though this is a well-known tool within rigid body simulation. Typically, objects are put to "sleep" based on when their linear and angular velocity drops below some threshold. This is trickier in practice than it sounds, and there can be serious problems in piling, as objects are repeatedly made to sleep and wake up when new objects are piled on. To get a sense of how our more aggressive piling-based sleep method compares to a deactivation method based strictly on velocity, we analyze the performance of the Bullet engine [15], which has an efficient implementation of object sleeping. Specifically, we compare Bullet with and without object sleeping, on the Bunny Matrix example (using the default physics). The difference obtained by allowing sleeping objects in Bullet is a 1.09x speedup on this example. This is comparable to, but somewhat lower than the speedups we see in our piling approach. Thus, we claim that our piling approach offers a moderate improvement in performance, compared to simple sleeping based on velocity.

5.3.2.3 Sleeping Threshold

The threshold, τ, used to determine when an object is put to sleep (see Section 5.2.3) has an impact on both the behavior and efficiency of the simulation. Table 5.2 shows timing results for Example 2, with varying values of τ, and the Figure 5.11 shows a visual comparison. Lower values of τ, indicating more aggressive sleeping, do yield a greater performance benefit. However, for very low values of τ, piles may

τ	none	0.7	0.6	0.5	0.4	0.3
Time(s)	495.2	412.3	269.8	231.9	215.6	199.4
Speed-Up	1.0x	1.2x	1.8x	2.1x	2.3x	2.5x

Table 5.2: Effect of sleeping threshold, τ, on performance of Example 2. Lower τ values give greater performance improvement, but very low values can cause unnatural-looking piles.

begin to look unnatural, as some objects are put to sleep before they are truly at rest. Very high values of τ may result in little piling action at all. In all examples in this chapter, we use $\tau = 0.5$, which provides a good performance improvement, while generating believable pile structures.

5.3.2.4 Error Measure

Measuring error in this sort of simulation is not straightforward. In particular, the ability to set an angle of repose, while producing more useful simulations, also creates "error" vs. a basic implementation that has a de-facto (and likely different) angle of repose. The examples we have tested (with $\tau = 0.5$) all look plausible, however this is a difficult feature to quantify.

In order to have some quantitative measure of the error our method introduces, we have developed an error metric based on relative motion of individual objects. Assume x_i^j is the position of (the center of) object i at frame j in the full simulation (without piling), and let $\tilde{x}_i^j$ denote the position in the simulation with piling. We calculate the frame-to-frame motion of the objects: $m_i^j = x_i^{j+1} - x_i^j$ (and similarly for $\tilde{m}_i^j$). The relative error is calculated as $e_i^j = |m_i^j - \tilde{m}_i^j|/m_i^j$, and the absolute error as $E_i^j = |m_i^j - \tilde{m}_i^j|$. These error values represent how different the frame-to-frame motion of the object is between the two simulations.

In interpreting these error measures, note that the individual e_i^j and E_i^j are unlikely to be helpful to consider in isolation, since a single particular object might behave quite differently between two simulations, although the simulation as a whole might be very similar. What is of more interest is the degree to which the change in simulation method creates a global change in the character of the simulation. This can be inferred by averaging the e_i^j or E_i^j over all i, which will give a measure of the overall amount of difference in motion between the two simulations; note that the absolute error roughly corresponds to average momentum, and thus does not have a meaningful scale in general. We report both relative and absolute error for two examples, shown per frame in Figures 5.9 and 5.10, and relative error averaged over all frames in Table 5.1.

In examining the error rates over the course of simulations, we see that the error rates fluctuate significantly. Notice that while the error in Examples 5 and 6 are approximately the same on average, they behave quite differently during the simulation. Note also (e.g. end of Figure 5.9) that the relative error becomes a less reliable measure as the simulation comes to a near-rest (the m_i^j denominators in e_i^j approaching 0). We notice that the peak errors in Figure 5.10 tend to occur at points of significant overall motion (i.e. when rings "splash" out of the box).

5.4 Conclusion and Discussions

Our approach for simulating piles of objects offers a simple method for representing one of the more interesting features of large rigid body simulations. By representing the piles explicitly, we can deactivate simulation of objects within the pile, enabling noticeable speedups. Designers of a rigid body simulation can achieve a desired object piling effect without implementing a more complex and slower frictional model component. In this way, we not only allow for a richer variety of simulations, but we achieve this with somewhat faster performance.

There are some clear limitations to our approach. If we simulate with multiple objects whose volume varies significantly (e.g. marbles and basketballs), the way a pile should form is difficult to specify, and our method would not perform well. Small variations (e.g. a factor of up to 2 or somewhat more) in volume should not be an issue, however. Also, our approach works more for random piles, rather than structured stacking (see Erleben [20] for an approach for structured stacking).

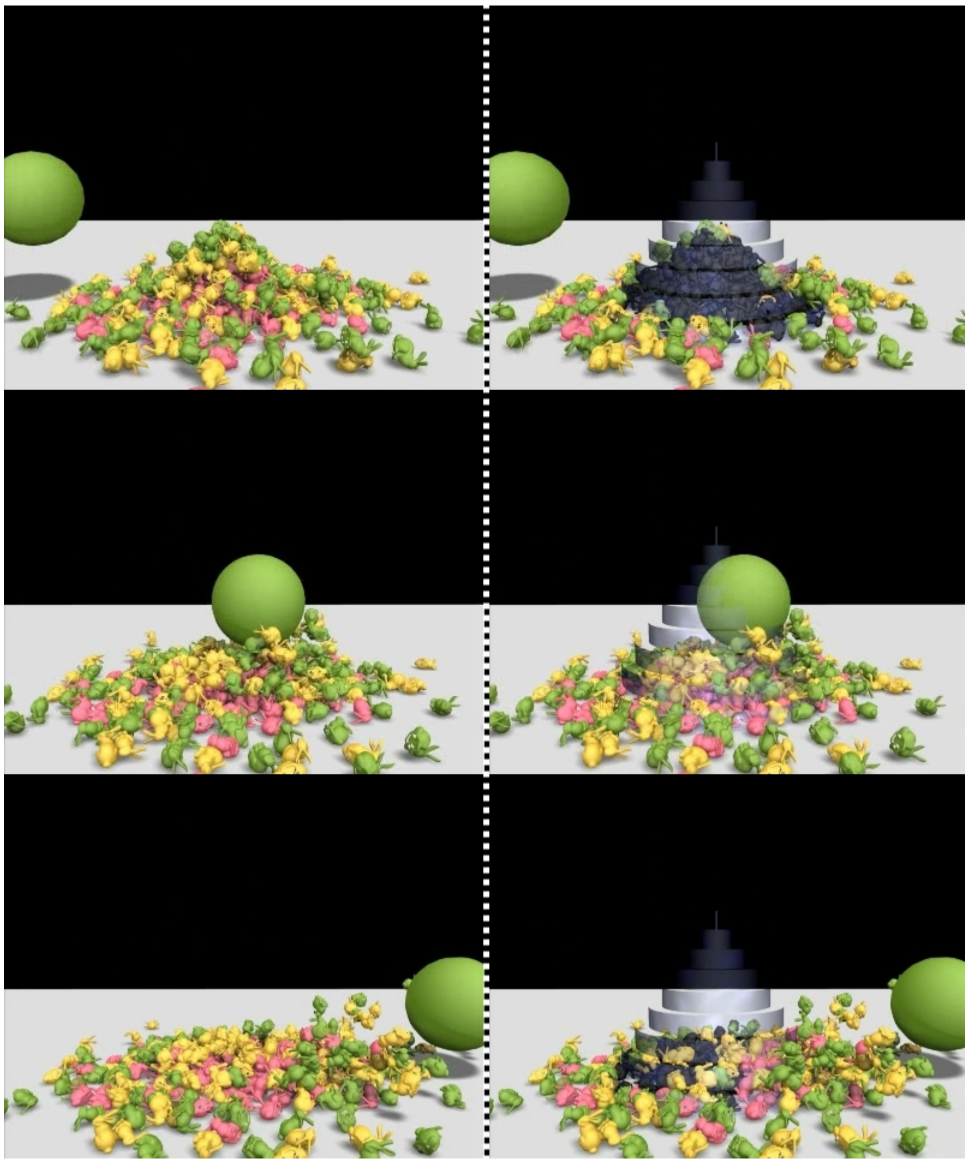


Figure 5.8: Example 9—A pile being broken up after being hit by a high momentum object.

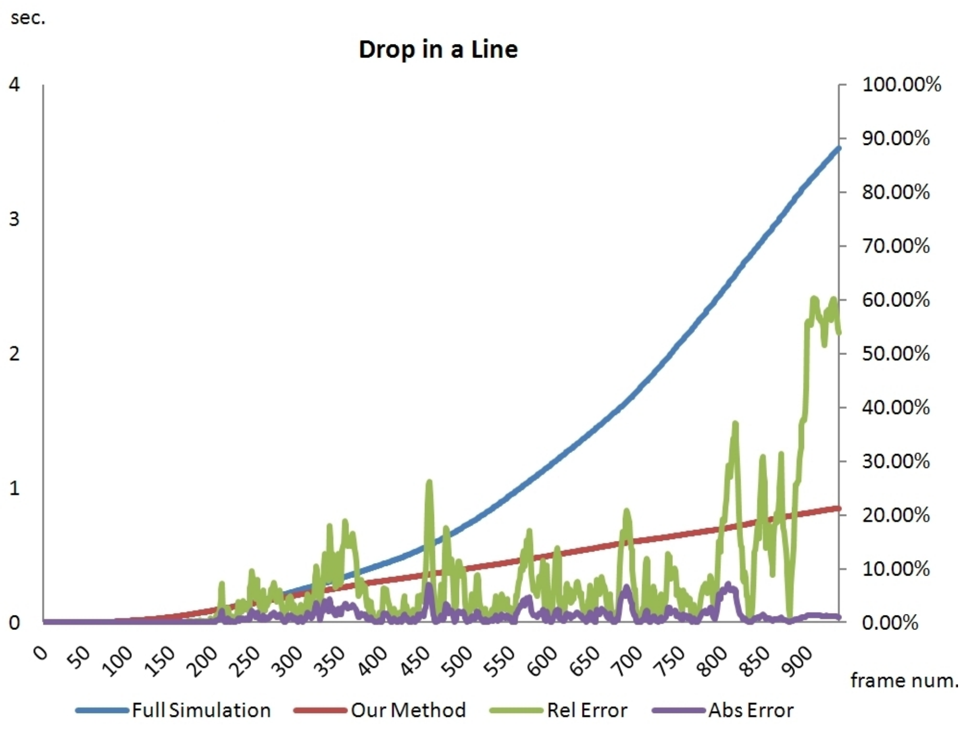


Figure 5.9: Timing and Error for Example 5—Drop in a Line. Left axis shows time per frame (in seconds), and right axis shows relative error. Scaled absolute error is superimposed.

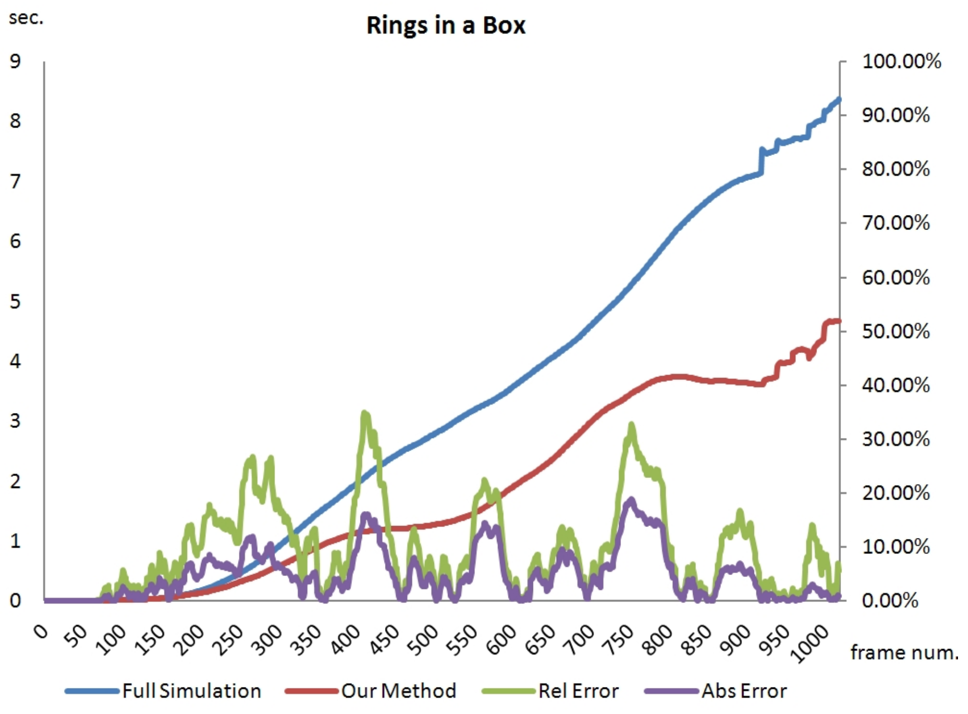


Figure 5.10: Timing and Error for Example 6—Rings in a Box. Left axis shows time per frame (in seconds), and right axis shows relative error. Scaled absolute error is superimposed.

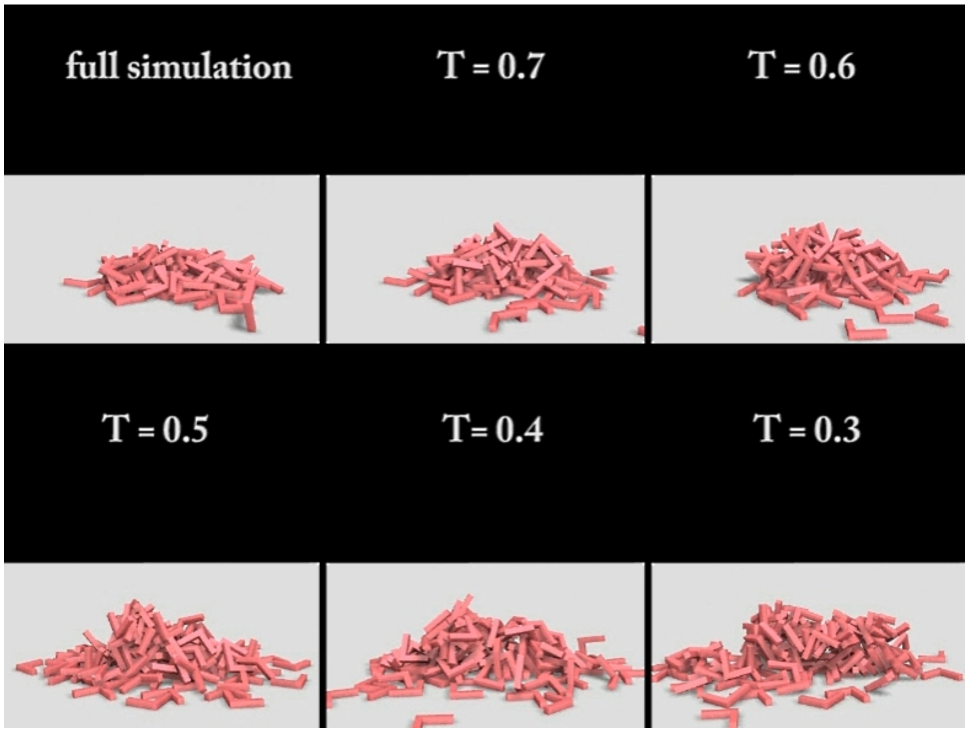


Figure 5.11: Comparison of sleeping thresholds

6. MODELING PILES OF OBJECTS VIA AUTOMATED CONSTRAINT PLACEMENT*

6.1 Introduction

In this chapter, we develop a more flexible method for modeling piles of objects. Modeling massive, disordered piles of objects, e.g. stone piles or food piles are used in the creation of digital content such as video games and films. Sometimes, the desired shapes of the piles might not be conical-like shapes. In the previous section, our pile modeling method is limited to conical shapes. Although we can place conical shapes to approximate more shapes, it is not very intuitive to set up the control.

To create appealing results and to satisfy specific needs of an artist, a goal-directable method is desired. We want a more flexible method such that users can provide initial designs of their desired piles, which could be polygon meshes or configurations for a set of objects, and the method will match simulation results to their initial stacking designs.

However, this goal-directable problem is very challenging because simulations are usually formulated as initial configuration problems. Given an initial configuration of a simulation scene, the simulator will evolve the initial configuration step by step. There is no guarantee that the results of the simulation will still be close to the design that a user provided. In addition, directly tweaking the simulation by adjusting the physics parameters is inefficient, and often ineffective, since we do not know the mapping between the parameters and our goal.

While procedural synthesis approaches are able to synthesize piles explicitly, the resulting piles cannot be directly used in a dynamic simulation since the synthesis

*Part of the data reported in this section is reprinted with permission from Automated Constraint Placement to Maintain Pile Shape by Shu-Wei Hsu and John Keyser, 2012. ACM Transactions on Graphics, Vol. 31, No. 6, Article 150, Copyright 2012 by ACM.

process mainly considers the geometry properties, not the dynamic simulation properties. If users want to use this resulting pile in a dynamic simulation further, e.g. stack more objects on it or destroy it with other objects, it is very possible the shape will fall as soon as dynamic simulations begin.

The main result of this section is a method for stabilizing the stacking process of simulations, helping preserve the design of object stacking. We demonstrate that our method is able to guide the stacking simulation to desired pile shapes. Plausibility of the simulation is also maintained. During simulation, our method eliminates the degrees of freedom (DOFs) of certain stacking objects which are considered "stable" locally. It helps preserve the volumes of the desired shapes. Our method includes two stages. The goal of the first stage is to find object candidates that satisfy static equilibrium conditions locally. In the second stage, we evaluate whether the DOFs of candidates can be removed to stabilize the structure without decreasing plausibility. Then we group eligible candidates to remove some DOFs temporarily, stabilizing the stacking structure.

6.2 Methodology

For achieving goal-directable massive stacking control, our idea is to strengthen a stacking structure by decreasing the degrees of freedom (DOFs) of objects during simulation. This helps reduce the perturbation in the structure and helps preserve the volume of the shape as well. In multibody simulation, generally each single object has six DOFs. The total DOFs of a simulation scene is proportional to the number of objects it has. During simulation, our method detects objects that are aggregated and reduces the DOFs of the scene by grouping those objects temporarily.

While this alone provides stabilization for stacking, it poses challenges for the fidelity of the simulation. As an extreme example, we could freeze every single

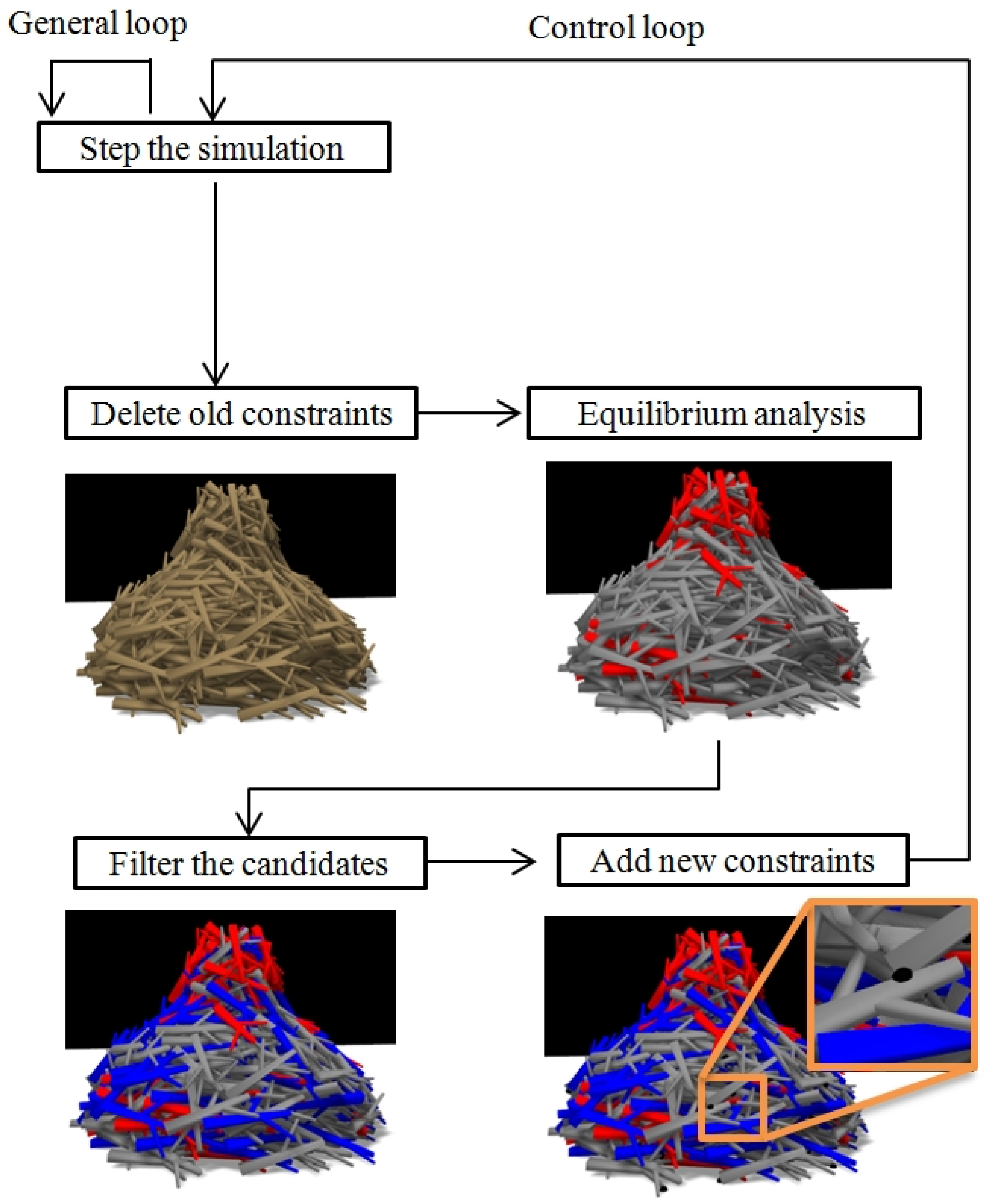


Figure 6.1: Control steps are taken every few simulation steps. First old constraints are removed, then local equilibrium analysis is performed (red objects are those not passing). Next candidates are filtered based on friction and contact point distribution requirements (blue objects are those not passing). Finally constraints are applied to some objects in the remaining candidates (see black dot in picture).

object in a stacking structure. The DOFs would then be zero. In this case, although it preserves the shape, it also causes unrealistic visual results since it does not consider whether an object is stable enough to stay at its current position or not. To maintain fidelity, we also need to examine whether an object is feasible for stacking before we try to group it.

Our method (see Figure 6.1) therefore includes two stages: 1) find locally stable objects, 2) manage the DOFs. In the first stage, our strategy is to use equilibrium analysis to find objects that are locally stable. In the second stage, we pick suitable candidates and insert kinematic constraints to group objects in a way that maintains the structure, and delete constraints to maintain the fidelity of the simulation. As the simulation is running, we periodically perform these two stages (every 20 time steps in our implementation). We give a brief algorithmic outline in Algorithm 1.

Algorithm 2: Simulation loop

```
i = 0;
while simulating do
    step simulation;
    if i++ % ctrlcycle == 0 then
        delete all constraints;
        for each object do
            perform local equilibrium analysis (Sec 3.2);
        end
        add constraints (Sec 3.3);
    end
end
```

Note that not all locally stable objects can be grouped. At the equilibrium analysis stage, we assume every single object is static (not moving). There exists a fuzzy gap between the result of equilibrium analysis in a static environment and the real situation in the dynamic simulation environment. Some candidates should not be grouped, e.g. an object which is close to an unstable situation. Adding constraints to those objects could lead to artifacts that may be quite noticeable. We filter out those objects by considering the gap between static friction and kinetic friction.

6.2.1 Local Equilibrium Analysis

In the first stage of our control method, we use equilibrium analysis in a local scheme to identify whether an object satisfies the equilibrium condition locally. In the second stage (see Sec.6.2.2), the analysis results, including the equilibrium condition and the solved forces, will be utilized to determine whether we can insert constraints to "nail" an object to its neighboring objects.

Local equilibrium analysis means we consider an object and all its contact points as an independent system and check whether the object is stable under that system. The formulation is same as for global equilibrium analysis but just handles one single object and its corresponding contacts.

First, we show how to compose the coefficient matrix of equilibrium equation, A_{eq}, for our local analysis scheme. Let $n+1$ be the number of contact points of an object. The dimension of A_{eq} is $6 \times 3(n+1)$.

$$\mathbf{A}_{eq} \cdot \mathbf{f} + \mathbf{w} = 0$$

$$\begin{bmatrix} \mathbf{A}_0 & \mathbf{A}_1 & \dots & \mathbf{A}_n \end{bmatrix} \begin{bmatrix} \mathbf{f}_0 \\ \mathbf{f}_1 \\ \vdots \\ \mathbf{f}_n \end{bmatrix} + \mathbf{w} = 0$$

$\mathbf{f}_i$ is an unknown sub-vector of dimension 3×1. Each $\mathbf{f}_i$ includes one pressure force, f_n, and two friction forces, f_{t_1}, and f_{t_2}, to be solved.

w is the natural mass property of the object when it is at rest. The dimension is 6×1. The first three components represent the weight vector which is (0, mg, 0). The last three components represent the torque vector which in this case is (0,0,0).

Each sub-matrix $\mathbf{A}_i$ stores the coefficients for computing the force and torque working on the contact point i. The dimension is 6×3.

$$\mathbf{A}_i = \begin{bmatrix} \hat{\mathbf{n}^i} & \hat{\mathbf{t}_1^i} & \hat{\mathbf{t}_2^i} \\ (\mathbf{n}^i \hat{\times} \mathbf{r}^i) & (\mathbf{t}_1^i \hat{\times} \mathbf{r}^i) & (\mathbf{t}_2^i \hat{\times} \mathbf{r}^i) \end{bmatrix}$$

where $\hat{\mathbf{n}^i}$ is the direction of the contact normal and $\hat{\mathbf{t}_1^i}$ and $\hat{\mathbf{t}_2^i}$ are two orthogonal tangent directions. $\mathbf{r}^i$ is the vector pointing from the center of mass of the object to the position of contact point i.

Next we show how to compose the coefficient matrix of friction conditions, A_{fr}. The dimension is $4(n+1) \times 3(n+1)$.

$$\mathbf{A}_{fr} \cdot \mathbf{f} \leq 0$$

$$\mathbf{A}_{fr} = \begin{bmatrix} \mathbf{H} & & 0 \\ & \ddots & \\ 0 & & \mathbf{H} \end{bmatrix}, \quad \mathbf{H} = \begin{bmatrix} -\mu_s & 1 & 0 \\ -\mu_s & -1 & 0 \\ -\mu_s & 0 & 1 \\ -\mu_s & 0 & -1 \end{bmatrix}$$

μ_s is the coefficient of static friction. Basically, each friction force needs two additional constraints to ensure it satisfies the friction pyramid model.

Note that local equilibrium does not yield global equilibrium, but there are three reasons that we choose to perform the equilibrium analysis locally.

1. The goal of this control method is not to put the structure into global equilib-

rium, but to maintain the original design in a result that looks like equilibrium. We observe that the local equilibrium analysis meets this requirement, especially in massive scenarios.

2. It provides more information than global analysis. Equilibrium analysis basically provides a yes-no answer. However, in simulation, there are often objects in motion. The global analysis may always return "no", which can be problematic for identifying objects that could be grouped. Instead, if we perform local analysis, objects that are already aggregated in a steady state will be revealed.

3. It is faster than global analysis. Since the time complexity of solving linear programming is super linear, solving multiple small systems is much faster than a single large system. In practice, we observe 2 to 3 orders of magnitude improvement on our examples.

Local analysis could overestimate the equilibrium situation (identify non-equilibrium as equilibrium). In Figure 6.2(a), object B is unstable in simulation. Selecting all contacts (p1 to p4) of B to do local equilibrium analysis gets a stable result of equilibrium so it adds constraints to freeze B. But it makes the structure unnatural. To avoid this, we only select contacts that can contribute positive forces in the Y (up) direction ($n_{cp} \cdot (0, 1, 0) > 0$). With this conservative selection strategy, the new equilibrium analysis for B uses p1 and p2, and it results in an unstable equilibrium since the torque cannot be kept to zero. The conservative selection strategy may underestimate (identify equilibrium as non-equilibrium) the equilibrium situation, but it doesn't cause artifacts. In Figure 6.2(b), object B1 is stable in simulation. Selecting partial contacts (p1 and p2) results in an unstable equilibrium (underestimation). No constraints will be added. Though B1 is underestimated, it doesn't cause artifacts. While selecting all points (overestimation) may cause noticeable artifacts,

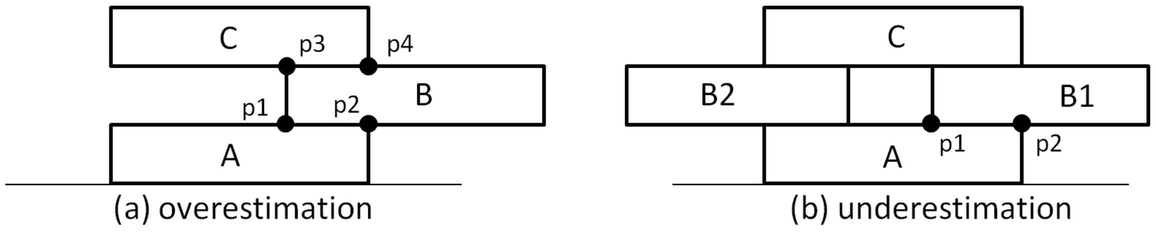


Figure 6.2: Contact selection strategies. (a) All-selecting. B is identify as equilibrium, which causes a visual error. (b) Conservatively-selecting. Though B1 is identify as non-equilibrium it doesn't cause visual errors.

conservative underestimation only limits potential speedups and is thus preferred.

Note that the contact points for analysis are the result of narrow phase collision detection. The simulator and the analysis module share the same copy of this information. However, the forces from the equilibrium analysis and the forces (or more specifically, the impulses) from the simulator are independent. We do not use the collision impulses to measure equilibrium since they are designed to avoid penetration rather than measure forces.

6.2.2 Constraints Management

We begin the second stage by examining the results provided by equilibrium analysis. This examination selects objects which are reasonable to be grouped. We then manage the insertion and deletion of constraints to strengthen the structure and maintain the fidelity of the simulation.

Grouping equilibrium objects can be problematic. We make an assumption that each object is at rest in the first stage. Instead, they are undergoing dynamic simulation so their status can be varied. We prefer not to group an equilibrium object if it is close to the boundary of non-equilibrium.

We want to discard objects that have large frictional forces. The concept that we use to select the threshold is the gap between static friction and kinetic friction.

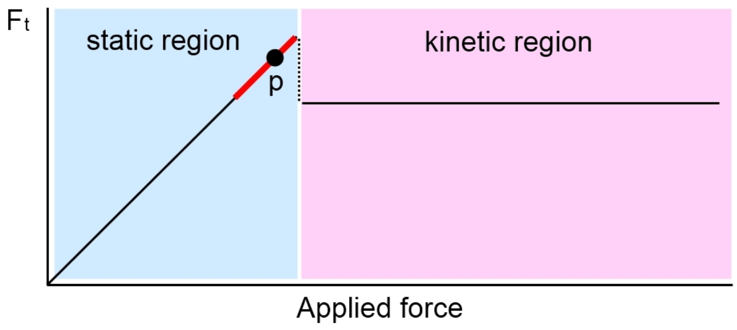


Figure 6.3: We set the kinetic friction as the maximal threshold for groupable objects. Any contact point of a groupable object cannot have a frictional force exceed this threshold.

We select objects that satisfy the following condition:

$$|F_{t_1}^i|, |F_{t_2}^i| < |\mu_k F_n^i| \qquad \forall i \in \text{contact points} \tag{6.1}$$

$F_{t_1}^i$ and $F_{t_2}^i$ are the tangential frictional forces, and F_n^i is the normal force. These are obtained from equilibrium analysis. μ_k is the coefficient of kinetic friction. We compute $|\mu_k F_n|$ as the maximal friction threshold. To explain why we set this threshold, we separate contact points into two categories, moving and at rest. Generally speaking, for moving contact points the maximal kinetic friction that the point can have is $\mu_k F_n$. If F_{t_1} or F_{t_2} is larger than the maximal kinetic friction, it means the object can not afford enough friction to maintain an equilibrium condition. For the second category, we know contact point i is at rest. According to a general friction profile (Figure 6.3), we consider the frictional force, F_t, is close to the moving point boundary if F_t is larger than the kinetic friction (even though it is in the static region). In this situation, it is very possible the contact will move soon hence we don't want to add a constraint to it.

The distribution of contact points is another issue that we have to consider for selecting groupable objects. To be a groupable object, we need two additional requirements:

1. Number of Contact Points. A groupable object should have at least three contact points, otherwise we will not add constraints to enforce its equilibrium status.

2. Distribution of Contact points. If the positions of all contact points are too close or are near a collinear situation, statistically it is not easy to be stable. We compute the following heuristic to detect this problem.

$$A = \sum_{i>2} \left| \frac{(p_i - p_{i-1}) \times (p_i - p_{i-2})}{2} \right| \qquad \forall i \in \text{contact points} \tag{6.2}$$

This sums the areas of triangles composed by three contact points with adjacent indices. If A is zero there is a collinear situation. If all contact points are close to each other, A will be relatively small, too. We set a threshold, α, so that if A is less than α we will not add constraints to enforce this object's equilibrium status. (α can be selected based on the object's size and shape). Note that this simple method filters out problematic objects more conservatively since a small value of A does not necessarily mean that all contacts are close to each other (e.g. if given four points, p_1 p_2 p_3, and p_4, with p_3 is extremely close to p_2, A will be close to 0 even if the four points form a large support in total).

Next, we start to reduce the DOFs by grouping objects. Denote the number of the candidate objects (i.e. those passing local equilibrium analysis, frictional limits, and contact point distribution/number limits) as n and the set of the candidate objects as $\mathbf{C} = \{c_i, i \in 1...n\}$. For each c_i, $\mathbf{S}^{c_i} = \{s_j^{c_i}, j \in 1...m\}$ denotes a set of surrounding objects that c_i is colliding with. We first check whether $\mathbf{S}^{c_i}$ is a subset of $\mathbf{C}$. If not, we consider the DOFs of c_i not ready to be reduced since the neighborhood is not fully stable. If yes, for each pair of objects, $p = (c_i, s_j^{c_i})$, we add one constraint between

param	description	value
μ_{sim}	coef. of friction in Bullet	0.7
μ_s	coef. of static friction (equilibrium analysis)	0.7
μ_k	coef. of kinetic friction (candidate selection)	0.6
α	collinear/support thres (geometry dependent)	0.5
κ	constraint break threshold	5

Table 6.1: Parameters and thresholds used in our implementation.

this pair, thus reducing the DOFs of the set. The constraint is placed at one of the contact points of p. Generally p has more than one contact point. We randomly pick one contact point as the position to add the constraint. This constraint limits all relative motion between c_i and $s_j^{c_i}$.

The specific type of kinematic constraint that we use is a modified slider constraint. A general slider constraint reduces 4 DOFs of two connected objects, allowing two objects to translate along one axis and rotate around the same axis relatively. We further force the translation distance to zero, making it similar to a nail pinning two objects together at the contact point. Other types of constraints could be used as well.

We have two additional settings for these constraints to maintain fidelity. First, the life cycle of those constraints are not permanent. In the beginning of the next control loop, all constraints will be deleted. Second, the constraint is breakable during simulation if the impulse applied on it exceeds a threshold κ (in our implementation, $\kappa = 5m$ where m is the mass of the object).

6.3 Implementation and Results

We use the Bullet open source physics library[15] as a base for implementing our control method. In default, the kinematic constraints in Bullet are not hard

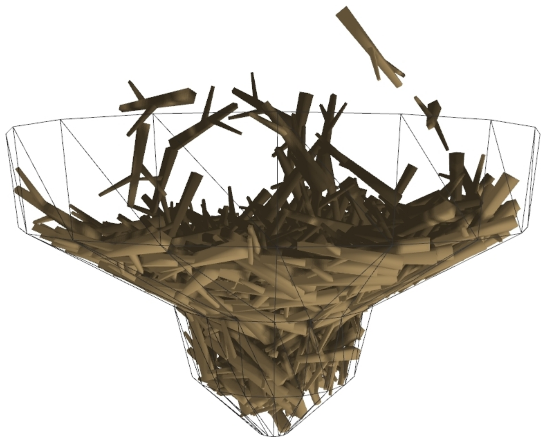


Figure 6.4: We use the upsidedown approach to generate an initial configuration set closing to the guiding mesh. Here is the guiding mesh in Example 2.

example	Cards	Sticks	Bananas	Books
number of objects	40	609	1008	879
constraints added	36	106	1072	630
simulation step (sec.)	0.0007	0.0519	0.2643	0.0180
analysis step (sec.)	0.0251	0.4697	3.3550	0.4042

Table 6.2: Performance from the average of last 512 steps.

constraints, so objects might shift a little bit even with constraints. To ease this problem, we increase the value of ERP (the joint error reduction parameter) in Bullet from 0 to 0.8 to make constraints harder. Linear programming is solved by using the Gurobi Optimizer solver [25]. All results are collected on a system with an i7 3.8GHz processor (using only one core), 16 GB of RAM, and an NVidia GTX580 graphics card. Step size is 1/240 second. One control step is run after every 20 simulation steps. Table 6.1 gives the parameters we use. Table 6.2 shows performance statistics. In our examples, a control step is roughly an order of magnitude slower than a general simulation step. Table 6.3 shows the timing breakdown of our Example 3. The overall effect of adding our control method is to approximately double the simulation time.

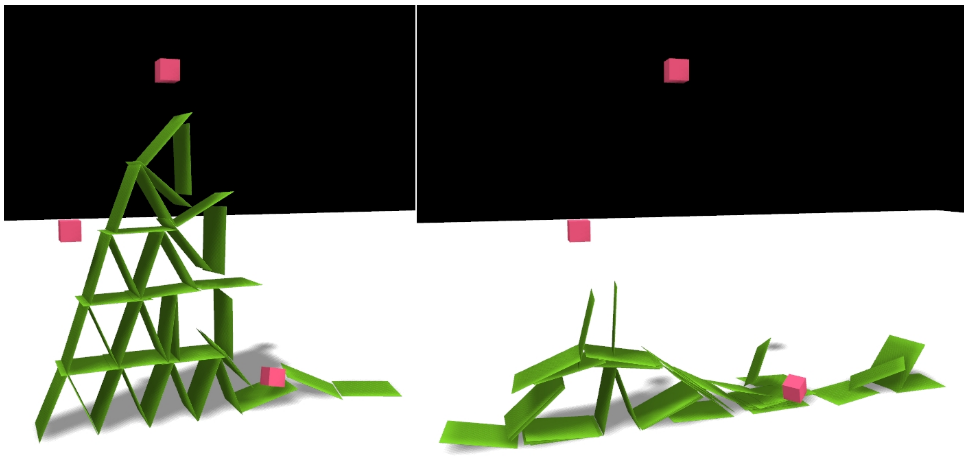


Figure 6.5: Left, simulation with our control method. The shape of the tower can be preserved until the dropped cubes hit it. Right, simulation only. The tower falls before any cube hits it.

loop	times(sec.)
one control loop	3.3
one simulation loop with constraints	0.26
one simulation only, no constraints	0.20

Table 6.3: The timing breakdown of Example 3, Bananas in a cart

6.3.1 Examples

We create four examples to demonstrate different scenarios where our method can help preserve stacking designs during simulation. We also test different types of objects as stacking elements, e.g. brick-like objects, thin slices, and nonconvex objects, in different examples.

Example 1, Card tower (Figure 6.5) In this example, we try to preserve an initial configuration of a stacking design. A card tower stands at the beginning of the simulation, and then we drop cubes one by one to break the tower. We want to maintain the tower structure until it is hit. Without our control method, the tower quickly falls under its own weight. Adding our control, the tower is stable until it is hit. For further comparison, we also try this example with larger friction coefficients but without using control. While larger friction allows the tower to remain stable before being hit, it also creates significant artifacts .

Example 2, Stick pile (Figure 6.6) In this example, we show an art-directable pile creation using our control method. We want to form a stick pile with a given shape. The idea is 1) create an initial configuration (positions and orientations of all objects) close to the desired shape, 2) release objects, and 3) use our control to strengthen this process. To create the initial configuration, we use an idea following that of Cho et al.[13]. It works in three steps: we turn the shape upside down, treat it as a container for collision detection, and drop objects into the mesh until full (Figure 6.4). This gives us (once it is turned back over) an initial configuration close to the results that we want, but not necessarily in a stable configuration, i.e. a straightforward simulation would generally cause the pile to collapse. We then slowly release objects from the bottom to the top of the shape, using our control method to strengthen it. The result is a pile that maintains the art-directed look, but still looks plausible and is still suitable for simulation.

Example 3, Bananas in a cart (Figure 6.8) An alternative approach to guide stacking is to directly drop objects into a region and apply our method in that region. This provides more flexibility than the "upsidedown" approach for designing an initial configuration set. In this example, we drop about 1,000 bananas into a cart to form a banana hill, which is not achievable by a simulation-only method. The shape of the banana hill is controlled by a mesh. Objects inside the mesh are controlled by our method, those outside are not.

Example 4, Book pile (Figure 6.7) This scenario combines examples 1 and 2. A spiral book pile is generated first and is preserved until a giant ball hits it.

The “upsidedown” approach is used to create the initial configuration set. After the structure is formed, we release a giant ball to crash into it. Without control, the structure collapses even before hit by a ball.

Example 5, Chair pile (Figure 6.9) In this example, we test our method with a chair model. Though the chair model has a higher concave shape than other models in our previous examples, our model is still able to handle it.

Example 6, Tire pile (Figure 6.10) In this example, we mimic an interesting and exaggerative tire pile form a photograph created by Alain Delorme[16]. To mimic this pile, we manually placed tires to copy the overall looking of our initial tire structure. With this initial pile as input, our method successfully stabilized this structure. Without using our method, the initial pile immediately fell.

Other examples Figure 6.11 demonstrates that our method can handle slight initial penetrations between the objects, which may happen in a manually-created structure. Figure 6.12 compares selection strategies (all vs. conservative - (Sec. 6.2.2)) for contacts to build the local equilibrium system. Artifacts appear without our selection strategy.

6.4 Conclusion and Discussions

We have presented a method for stabilizing stacking behavior in dynamics simulations, to support art-directable stacking designs. An additional advantage of our method is that it can be easily added to existing solvers without modifying the solver fundamentally.

Our approach has some limitations. First, unlike optimization-based methods, our approach cannot *precisely* specify the ending configuration. It is also possible to direct to a shape that could never be supported stably, and thus our method would not support. Instead, our method is more suitable for *guiding* the simulation to maintain a given shape. Though this limits the range of piles it allows, it also helps ensure that the piles look physically plausible.

Another issue (that can be seen as a limitation or an advantage) is that the pile structure created is still subject to physics simulation. Thus, dropping objects at too high of a speed can add too much energy and break apart any pile that may have already formed. For example, if the bananas in Example 3 are dropped into the cart at too high of a speed, there will be no opportunity for a pile to form. While the user still has a great deal of flexibility in designing the shape, our control process can only help in maintaining or achieving a shape, it cannot completely negate physics.

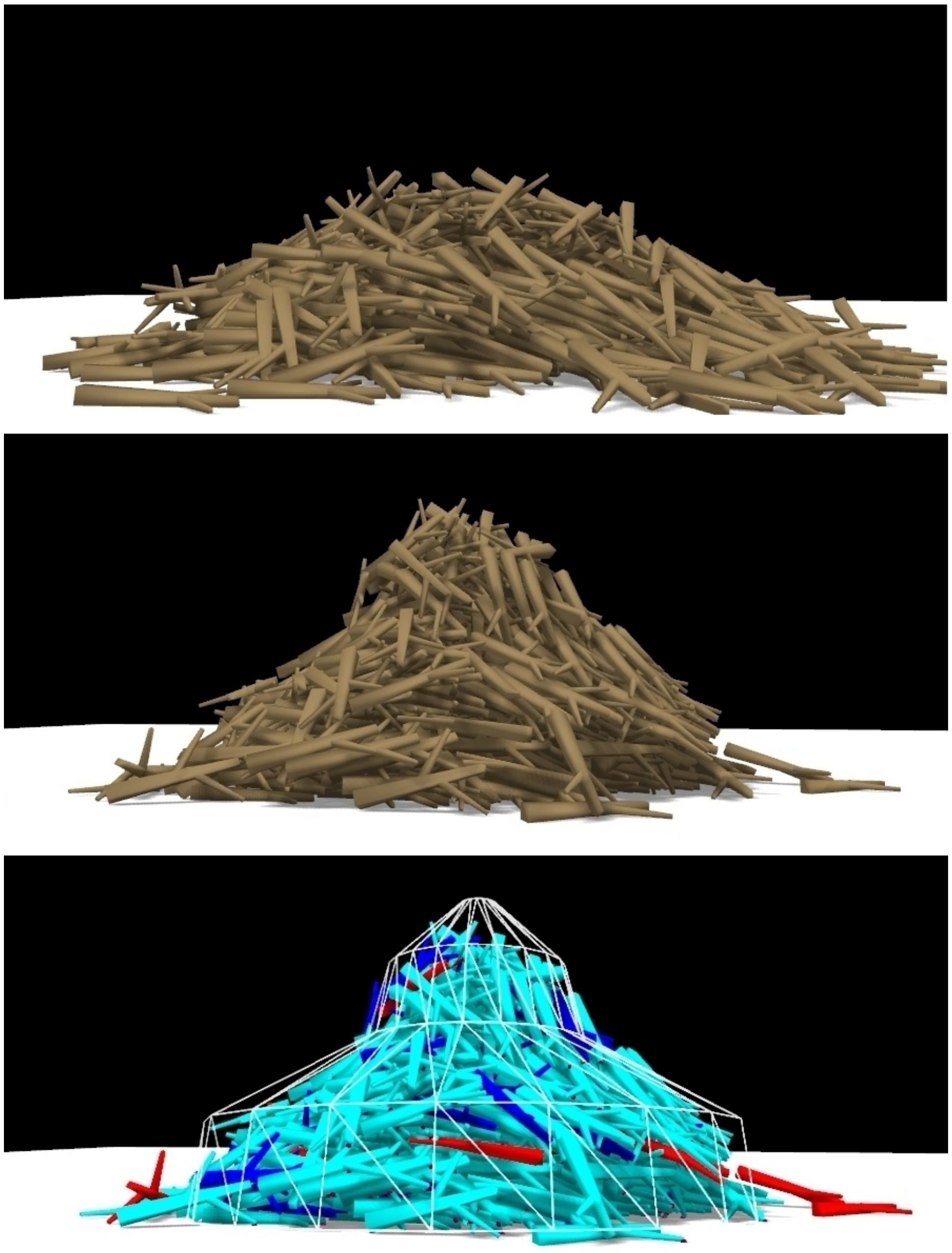


Figure 6.6: To create a pile of a given shape, a user specifies a desired shape and starting configuration of objects. At right, a pile starting in a desired configuration will collapse under simulation. The middle shows the result of adding constraints judiciously to maintain the pile shape. At right is the desired control mesh (in white), with the objects that have had constraints applied colored in cyan.

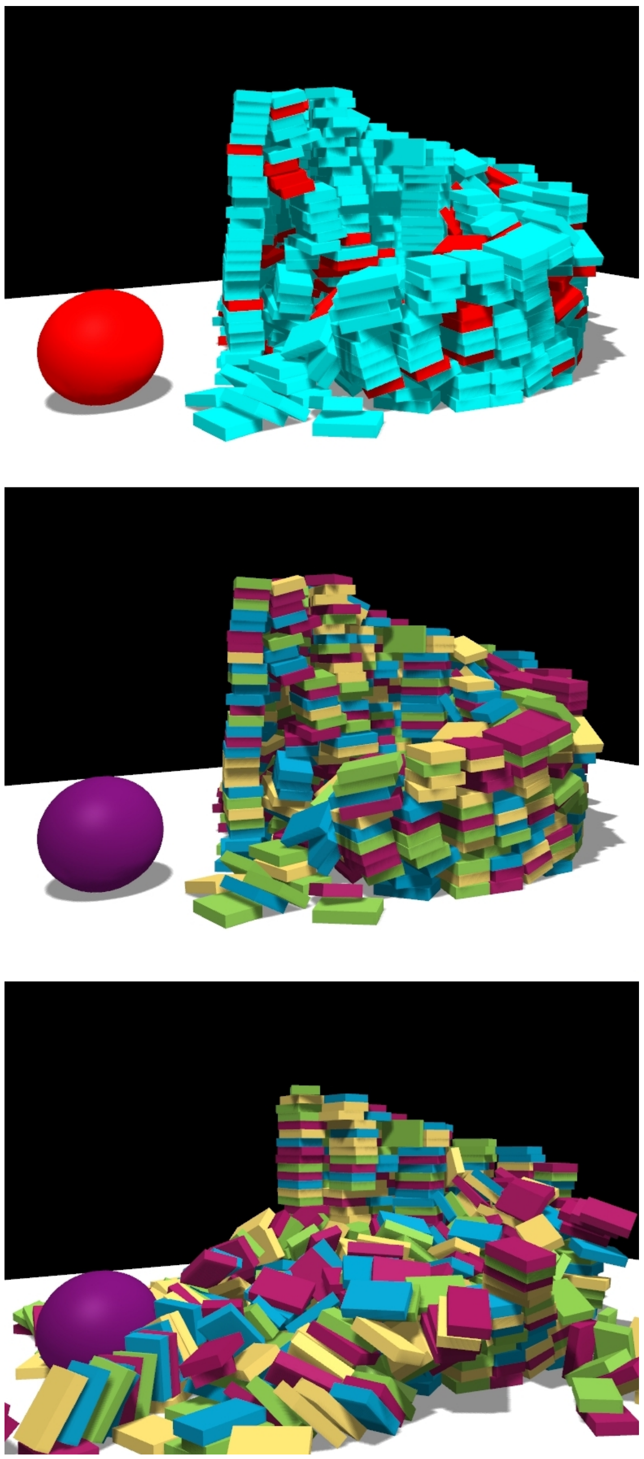


Figure 6.7: A pile of books. Top: The analysis result of current frame (cyan: stable objects; red: unstable objects). Middle: The simulation result with control. Bottom: The simulation result without control.

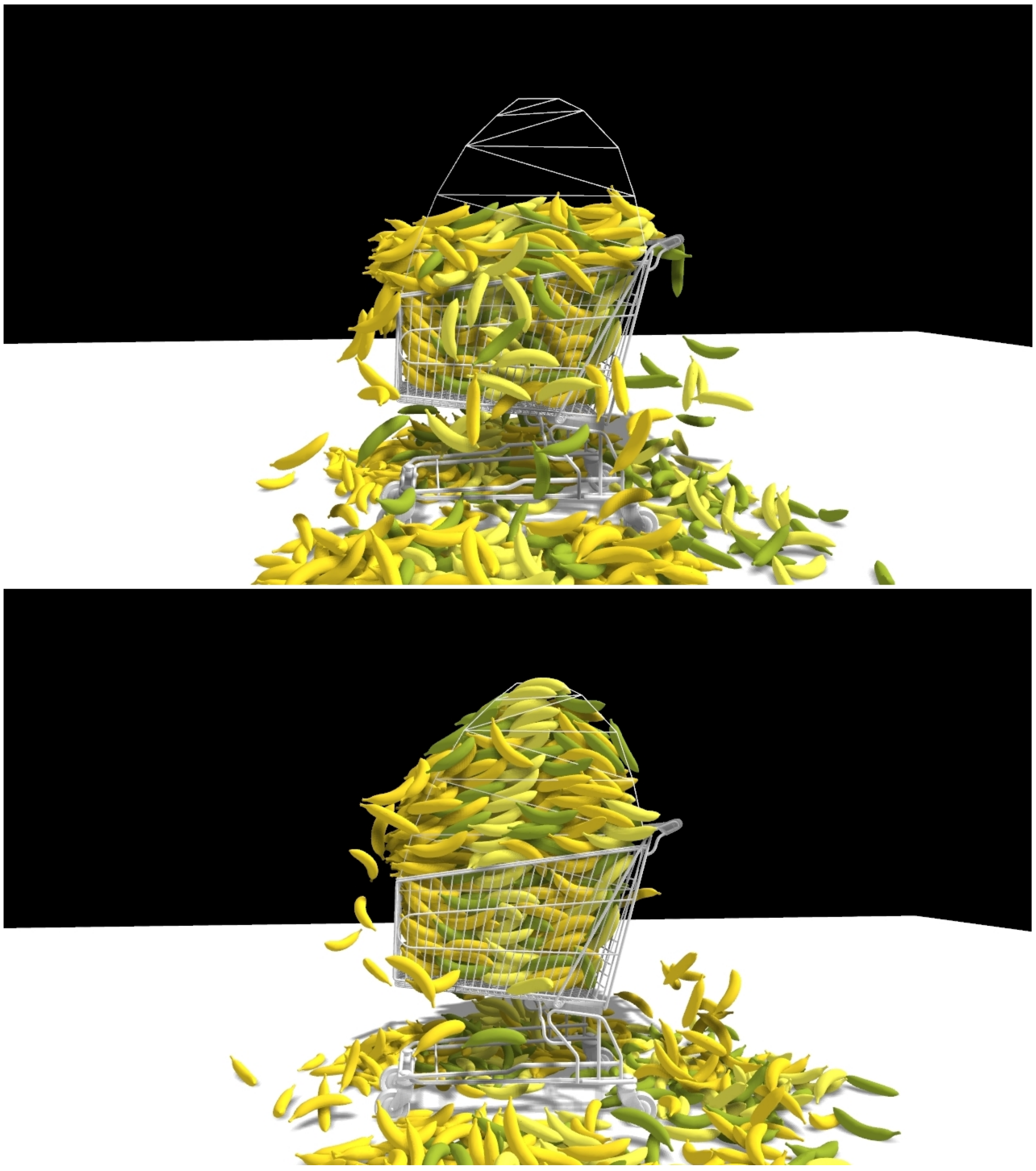


Figure 6.8: Bananas. At top, without our control method, the bananas exhibit only minimal piling. At bottom, our control method allows the banana pile to closely match the desired shape.

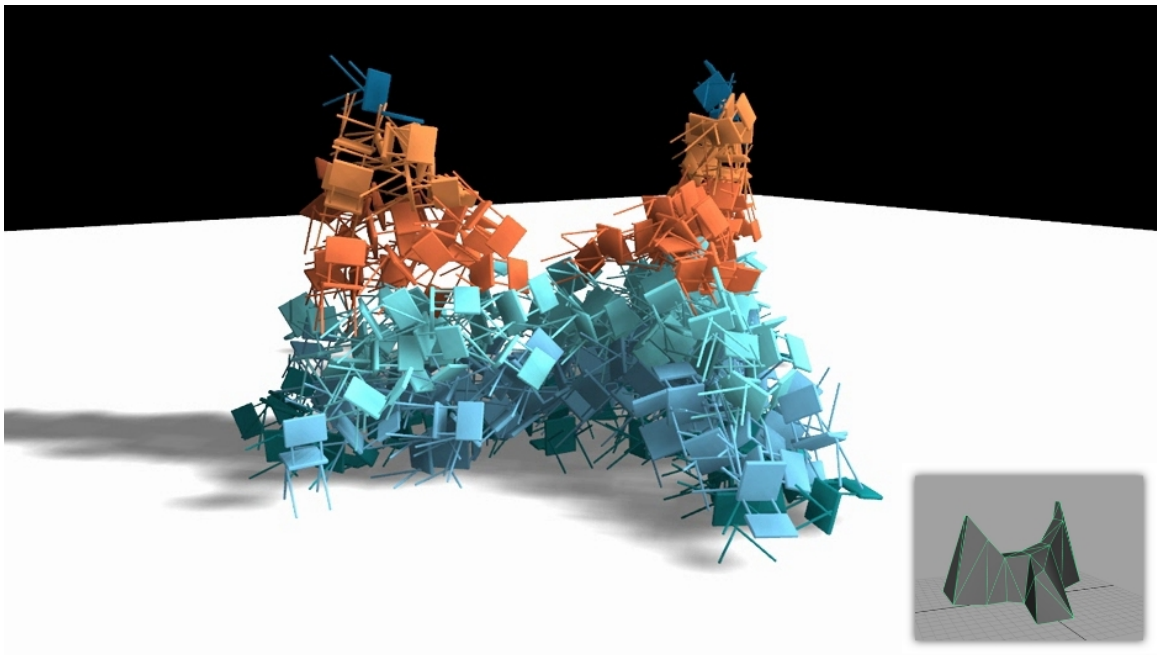


Figure 6.9: A chair pile created by our method. The mesh in the bottom right is our desired shape.

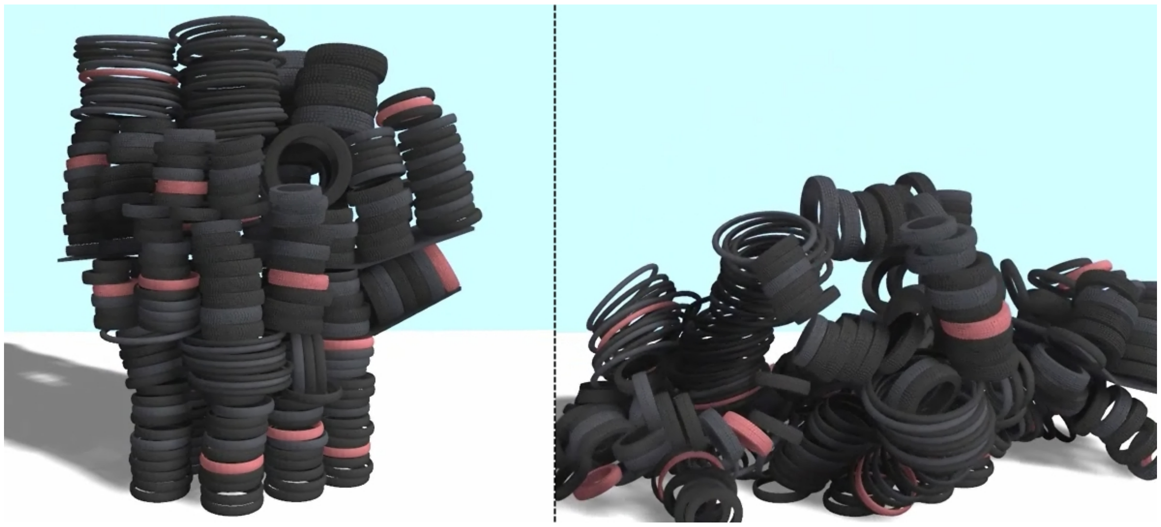


Figure 6.10: An exaggerative pile created by our method (the left image). We manually placed tires to create an initial pile and then use our method to stabilize the structure. Without using our method, the pile fell immediately (the right image)

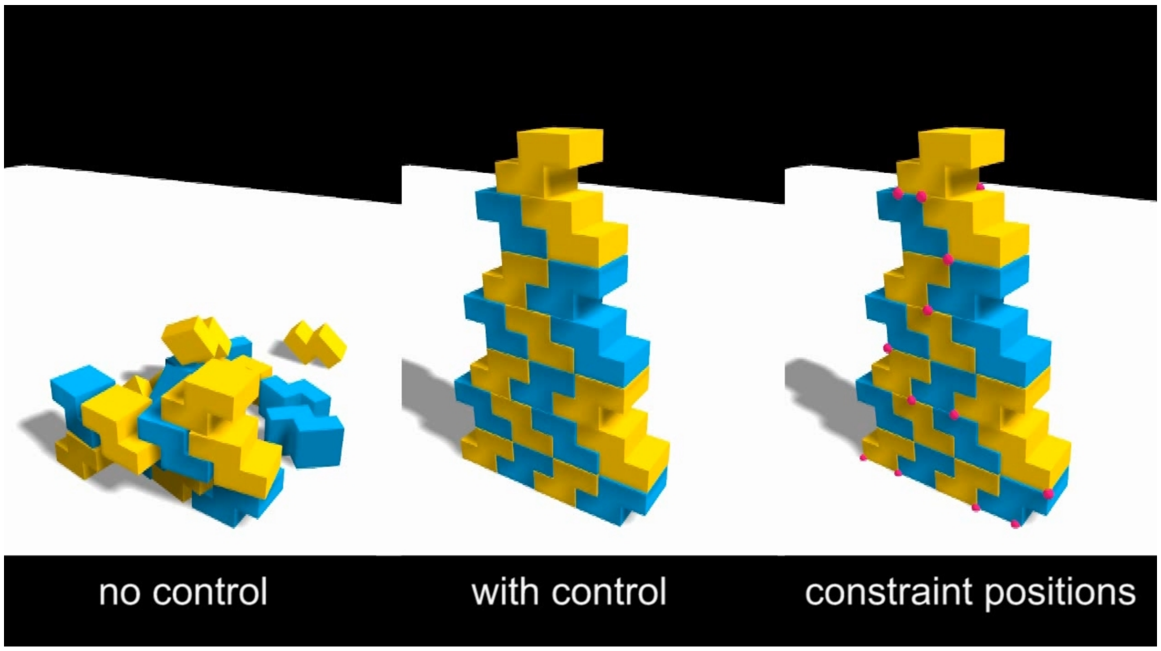


Figure 6.11: Our method can handle slight initial penetrations between the objects, which may happen in a manually-created structure. The left image is the result without using our control method. The center image is the result with our control. The right image shows the constraints positions of the controlled result.

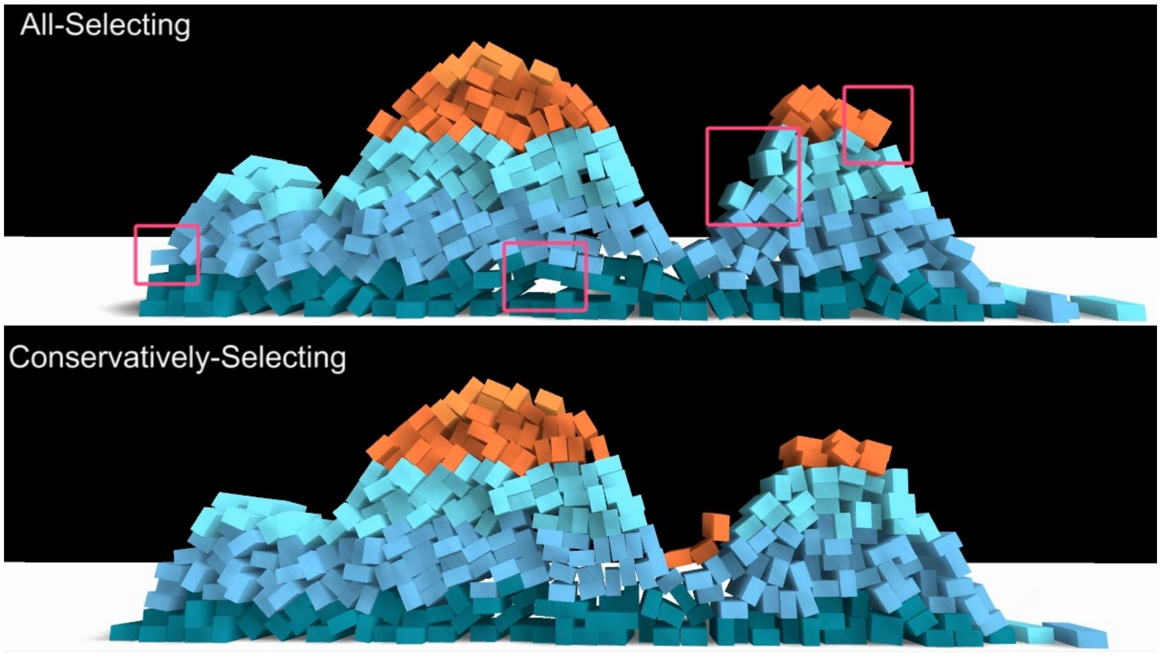


Figure 6.12: Comparison of contact selection strategies (all vs. conservative) for contacts to build the local equilibrium system. Artifacts appear without using our selection strategy.

7. CONCLUSION

This dissertation describes several techniques for efficiently simulating large numbers of rigid objects and for artistically guiding the piling phenomenon appearing in the large scale rigid body simulations. To conclude this dissertation, we first review the thesis statement of this dissertation and list the results for supporting this statement. Then, we summarize some limitations of our techniques. Finally, we discuss future research of this work.

7.1 Summary of Results

Again, the thesis statement of this dissertation is:

Simulating large numbers of rigid bodies can be made efficient by modeling the problem with a statistical computation scheme, and be made art-directable for stacking behavior through degree-of-freedom reduction techniques.

We prove this thesis statement by demonstrating the following things:

- Present a fast rigid body simulator using statistical techniques. This work demonstrates that our statistically-based collision approach can provide plausible results and major performance improvement.
- Extend the statistical method to a mixed method that resolve the collision according to an object's visual importance. Under this framework, we can easily handle a non-uniform simplification, creating plausible rigid body simulations and achieving better performance as well.
- Introduce two techniques to offer artistic control for modeling model the large

scale stacking phenomenon. The main idea behind these two techniques is properly reducing the DOFs. To do this properly, one utilizes the idea of angle of repose, and the other uses equilibrium analysis. These two techniques both allow us to specify the shapes of piles that can be obtained, allowing for a more realistic or more art-directed look to emerge.

7.2 Limitation

Here, we summarize the limitation of each technique we proposed. More detail about them can be found at the end of chapters 3, 4, 5 and 6.

For the statistical rigid body simulator:

- The method is appropriate for rigid body simulation situations requiring significant performance improvement, and allowing for some loss in fidelity.
- The statistical method has only limited (though still noticeable) performance improvement over a competing ultra-low resolution model.

For the mixed rigid body simulator:

- Again, due to the possible errors introduced by the simplified simulation, this method would be inappropriate in a setting in which high accuracy is desired across the entire scene.

For the first pile modeling techniques that use the idea of angles of repose:

- If we simulate with multiple objects, their volume cannot vary too much.
- This approach works more for random piles, rather than structured stacking.

For the second pile modeling techniques that use the idea of equilibrium analysis:

- The shape of the desired pile should still be physically plausible; otherwise it won't work.

7.3 Future Work

We have described several novel techniques for advancing large scale rigid body simulation. There are a number of ways that these techniques can spur further work in the future. We list some interesting directions here.

First, the statistically-based simulation model we have proposed in the dissertation could certainly be augmented in various ways, including determining an "optimal" sampling of the domain during signature gathering, or better handling of highly non-convex shapes. Not only will the sampling time decrease but also the quality of statistical signatures will increase.

Also, we believe that a statistical approach as described here has potential wider application, such as to deformable object simulation or to particle-based fluid simulation (other statistical approaches have already been applied to fluid simulation [67, 74]). Additional statistical behavior should be made to achieve this; for example, the statistical behavior of the local deformation due to the collision forces and inner forces. We expect to see more speedup by implementing these simulation systems. Moreover, the work presented in this dissertation discusses determination of the signature from an impulse-based simulation. However, it may be possible to collect signatures via other means, including measurement of real-world objects. Our work has demonstrated that a statistical model of collision response is sufficient to produce feasible results. This, then, opens up much wider simulation possibilities, including incorporation of real-world and not just virtual objects.

Finally, there are a couple exciting topics that can be extended from our pile modeling techniques. First, it would be interesting to fit our art-guiding techniques into articulated object simulation or deformable object simulation such as Position-Based Dynamics[52]. Just like rigid objects, stacking phenomena often happen with these two types of objects. Fundamentally, there is no reason why it could not be extended to handle them, as well. Also, it would be very interesting to study the perceptual tolerance for this type of simulation controls. It might be useful to maximize the controllability level without breaking the plausibility of the simulation.

REFERENCES

[1] Tomas Akenine-Möller, Eric Haines, and Natty Hoffman. *Real-Time Rendering 3rd Edition.* A. K. Peters, Ltd., Natick, MA, USA, 2008.

[2] Yalmar P. Atencio, Claudio Esperanca, Paulo R. Cavalcanti, and Antonio Oliveira. A collision detection and response scheme for simplified physically based animation. In *Proceedings of SIBGRAPI, the 18th Brazilian Symposium on Computer Graphics and Image Processing*, pages 291–298, Natal, Brazil, 2005.

[3] David Baraff. Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, SIGGRAPH '94, Orlando, FL, USA, 1994.

[4] David Baraff. An introduction to physically based modeling: Rigid body simulation i - unconstrained rigid body dynamics. In *An Introduction to Physically Based Modelling, SIGGRAPH '97 Course Notes*, Los Angeles, CA, USA, 1997.

[5] Ronen Barzel, John F. Hughes, and Daniel N. Wood. Plausible motion simulation for computer graphics animation. In *Proceedings Eurographics Workshop Computer Animation and Simulation*, Poitiers, France, 1996.

[6] Jacob Beaudoin and John Keyser. Simulation levels of detail for plant motion. In *Proceedings ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, pages 297–304, Grenoble, France, 2004.

[7] Jan Bender, Kenny Erleben, Jeff Trinkle, and Erwin Coumans. Interactive simulation of rigid body dynamics in computer graphics. In *EUROGRAPH-*

ICS 2012 State of the Art Reports, Cagliari, Sardinia, Italy, 2012. Eurographics Association.

[8] Roman Berka. Reduction of computations in physics-based animation using level of detail. In *13th Spring Conference on Computer Graphics*, Budmerice, Slovakia, 1997.

[9] Deborah A. Carlson and Jessica K. Hodgins. Simulation levels of detail for real-time animation. In *Proceedings of the conference on Graphics interface '97*, Toronto, Canada, 1997.

[10] Stephen Chenney and D. A. Forsyth. Sampling plausible solutions to multi-body constraint problems. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '00, New Orleans, LA, USA, 2000.

[11] Stephen Chenney and David Forsyth. View-dependent culling of dynamic systems in virtual environments. In *Proceedings of Symposium on Interactive 3D Graphics*, pages 55–58, Providence, RI, USA, 1997.

[12] Stephen Chenney, Jeffrey Ichnowski, and D. Forsyth. Dynamics modeling and culling. *IEEE CGA*, pages 79–87, March/April 1999.

[13] Han Cho, Athena Xenakis, Erin Tomson, Stefan Gronsky, and Apurva Shah. Course6: Anyone can cook: Inside ratatouille's kitchen. ACM SIGGRAPH 2007 courses. San Diego, CA, USA, 2007.

[14] Jonathan D. Cohen, Ming C. Lin, Dinesh Manocha, and Madhav Ponamgi. I-collide: an interactive and exact collision detection system for large-scale en-

vironments. In *Proceedings of the Symposium on Interactive 3D graphics*, I3D '95, Monterey, CA, USA, 1995.

[15] Erwin Coumans et al. Bullet physics library, 2013. : bulletphysics.org.

[16] Alain Delorme, 2012. : http://www.alaindelorme.com/.

[17] Evan Drumwright. A fast and stable penalty method for rigid body simulation. *IEEE Transactions on Visualization and Computer Graphics*, 14(1):231–240, January 2008.

[18] Stephen A. Ehmann and Ming C. Lin. Accurate and fast proximity queries between polyhedra using convex surface decomposition. In *Proceedings of Eurographics '01*, pages 500–510, UK, 2001.

[19] L. Endo, C. Morimoto, and A. Fabris. Real-time animation of underbrush. In *Proceedings of 11th International Conference in Central Europe on Computer Graphics, Visualization, and Computer Vision*, Plzen Bory, Czech Republic, 2003.

[20] Kenny Erleben. Velocity-based shock propagation for multibody dynamics animation. *ACM Transaction on Graphics*, 26(2):12, 2007.

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

[22] Paul Fearing. Computer modelling of fallen snow. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques (SIGGRAPH '00)*, SIGGRAPH '00, pages 37–46, New Orleans, LA, USA, 2000.

[23] Scott Le Grand. *Broad-Phase Collision Detection with CUDA*. Addison-Wesley Professional, MA, USA, 2007.

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

[25] Gurobi Optimization, 2013. : http://www.gurobi.com.

[26] Takahiro Harada. A parallel constraint solver for a rigid body simulation. In *SIGGRAPH Asia 2011 Sketches*, SA '11, Hong Kong, China, 2011.

[27] Jacques Heyman. *The Stone Skeleton: Structural Engineering of Masonry Architecture*. Cambridge University Press, Cambridge, England, 1997.

[28] Shu-Wei Hsu and John Keyser. Statistical simulation of rigid bodies. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, New Orleans, LA, USA, 2009.

[29] Shu-Wei Hsu and John Keyser. Piles of objects. *ACM Transaction on Graphics*, 29(6):155:1–155:6, December 2010.

[30] Shu-Wei Hsu and John Keyser. Automated constraint placement to maintain pile shape. *ACM Transaction on Graphics*, 31(6):150:1–150:6, November 2012.

[31] Ruoguan Huang, Zeki Melek, and John Keyser. Preview-based sampling for controlling gaseous simulations. In *Proceedings of the 2011 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '11, Vancouver, Canada, 2011.

[32] Philip M. Hubbard. Approximating polyhedra with spheres for time-critical collision detection. *ACM Transaction on Graphics*, 15(3):179–210, 1996.

[33] Thomas Jakobsen. Advanced character physics. Technical report, http://www.gotoandplay.it/_articles/2005/08/advCharPhysics.php, 2005.

[34] Doug L. James and Kayvon Fatahalian. Precomputing interactive dynamic deformable scenes. *ACM Transactions on Graphics*, 22:879–887, 2003.

[35] Doug L. James and Dinesh K. Pai. Bd-tree: Output-sensitive collision detection for reduced deformable models. *ACM Transaction on Graphics*, 23(3):393–398, 2004.

[36] P. Jimenez, F. Thomas, and C. Torras. 3d collision detection: A survey. *Computers and Graphics*, 25:269–285, 2001.

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

[38] Danny M. Kaufman, Shinjiro Sueda, Doug L. James, and Dinesh K. Pai. Staggered projections for frictional contact in multibody systems. *ACM Transaction on Graphics*, 27(5):164:1–164:11, December 2008.

[39] Christian Lauterbach, Michael Garl, Shubhabrata Sengupta, David Luebke, and Dinesh Manocha. Fast bvh construction on gpus. In *Proceedings of Eurographics '09*, Garching, Germany, 2009.

[40] E. T. Y. Lee. Choosing nodes in parametric curve interpolation. *Computer-Aided Design*, 21(6):363–370, July 1989.

[41] Ming C. Lin and John F. Canny. A fast algorithm for incremental distance calculation. In *IEEE International Conference on Robotics and Automation*, pages 1008–1014, Sacramento, CA, USA, 1991.

[42] Fuchang Liu, Takahiro Harada, Youngeun Lee, and Young J. Kim. Real-time collision culling of a million bodies on graphics processing units. *ACM Trans. Graph.*, 29(6):154:1–154:8, December 2010.

[43] Lun Liu and Ling Zhou. Numerical study on sandpile formation of granular materials with different grain size distributions. *Geotechnical Engineering for Disaster Mitigation and Rehabilitation*, pages 374–380, 2008.

[44] R. K. Livesley. Limit analysis of structures formed from rigid blocks. *International Journal for Numerical Methods in Engineering*, 12:1853–1871, 1978.

[45] David Luebke, Benjamin Watson, Jonathan D. Cohen, Martin Reddy, and Amitabh Varshney. *Level of Detail for 3D Graphics.* Elsevier Science Inc., New York, NY, USA, 2002.

[46] Chongyang Ma, Li-Yi Wei, and Xin Tong. Discrete element textures. *ACM Transaction on Graphics*, 30(4):62:1–62:10, August 2011.

[47] Sebastian Martin, Bernhard Thomaszewski, Eitan Grinspun, and Markus Gross. Example-based elastic materials. *ACM Transaction on Graphics*, 30(4):72:1–72:8, July 2011.

[48] Raju Mattikalli, David Baraff, and Pradeep Khosla. Finding all gravitationally stable orientations of assemblies. In *Proceedings of 1994 IEEE International Conference on Robotics and Automation*, pages 251–257, San Diego, CA, USA, 1994.

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

[50] Brian Mirtich. *Impulse-based Dynamic Simulation of Rigid Body Systems.* University of California, Berkeley, CA, USA, 1996.

[51] Matthew Moore and Jane Wilhelms. Collision detection and response for computer animation. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '88, Atlanta, GA, USA, 1988.

[52] Matthias Müller, Bruno Heidelberger, Marcus Hennix, and John Ratcliff. Position based dynamics. *Journal of Visual Communication and Image Representation*, 18(2):109–118, 2007.

[53] Michael B. Nielsen and Robert Bridson. Guide shapes for high resolution naturalistic liquid simulation. *ACM Transaction on Graphics*, 30(4):83:1–83:8, July 2011.

[54] Michael B. Nielsen and Brian B. Christensen. Improved Variational Guiding of Smoke Animations. *Computer Graphics Forum*, 29(2):705–712, 2010.

[55] Michael B. Nielsen, Brian B. Christensen, Nafees Bin Zafar, Doug Roble, and Ken Museth. Guiding of smoke animations through variational coupling of simulations at different resolutions. In *Proceedings of the 2009 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*, SCA '09, 2009.

[56] David O'Brien, Susan Fisher, and Ming Lin. Automatic simplification of particle system dynamics. In *Proceedings of IEEE International Conference on Computer Animation*, pages 210–219, Seoul, Korea, 2001.

[57] Carol O'Sullivan and John Dingliana. Collisions and perception. *ACM Transaction on Graphics*, 20(3):151–168, 2001.

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

[59] Adrien Peytavie, Eric Galin, Stephane Merillou, and Jerome Grosjean. Procedural Generation of Rock Piles Using Aperiodic Tiling. *Computer Graphics Forum*, 28(7):1801–1810, 2009.

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

[61] Jovan Popović, Steven M. Seitz, Michael Erdmann, Zoran Popović, and Andrew Witkin. Interactive manipulation of rigid body simulations. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*, pages 209–217, New Orleans, LA, USA, 2000.

[62] Thiti Rungcharoenpaisal and Pizzanu Kanongchaiyos. A collision detection method for high resolution objects using tessellation unit on gpu. In *Proceedings of ACM SIGGRAPH 2012 Posters Section*, SIGGRAPH '12, Los Angeles, CA, USA, 2012.

[63] Harald Schmidl and Victor J. Milenkovic. A fast impulsive contact suite for rigid body simulation. *IEEE Transactions on Visualization and Computer Graphics*, page 2004, 2004.

[64] Min Tang, Dinesh Manocha, Jiang Lin, and Ruofeng Tong. Collision-streams: Fast gpu-based collision detection for deformable models. In *Symposium on Interactive 3D Graphics and Games*, I3D '11, Florence, SC, USA, 2011. ACM.

[65] Richard Tonge, Feodor Benevolenski, and Andrey Voroshilov. Mass splitting for jitter-free parallel rigid body simulation. *ACM Transaction on Graphics*, 31(4):105:1–105:8, July 2012.

[66] Daniel J. Tracy, Samuel R. Buss, and Bryan M. Woods. Efficient large-scale sweep and prune methods with aabb insertion and removal. In *Proceedings of the 2009 IEEE Virtual Reality Conference*, VR '09, LA, USA, 2009.

[67] Adrien Treuille, Andrew Lewis, and Zoran Popović. Model reduction for real-time fluids. *ACM Transaction on Graphics*, 25(3):826–834, 2006.

[68] Christopher D. Twigg and Doug L. James. Many-worlds browsing for control of multibody dynamics. *ACM Transaction on Graphics*, 26(3), July 2007.

[69] Christopher D. Twigg and Doug L. James. Backward steps in rigid body simulation. *ACM Transaction on Graphics*, 27(3):25:1–25:10, August 2008.

[70] Kelly Ward, Ming Lin, Joohi Lee, Susan Fisher, and Dean Macri. Modeling hair using level-of-detail representations. In *Proceedings Of Computer Animation and Social Agents*, pages 210–219, New-Brunswick, NJ, USA, 2003.

[71] Emily Whiting, John Ochsendorf, and Frédo Durand. Procedural modeling of structurally-sound masonry buildings. *ACM Transaction on Graphics*, 28(5):112:1–112:9, December 2009.

[72] Emily Whiting, Hijung Shin, Robert Wang, John Ochsendorf, and Frédo Durand. Structural optimization of 3d masonry buildings. *ACM Transaction on Graphics*, 31(6):159:1–159:11, November 2012.

[73] Emily Jing Wei Whiting. *PhD Thesis: Design of structurally-sound masonry buildings using 3D static analysis*. Massachusetts Institute of Technology, MA, USA, 2012.

[74] Martin Wicke, Matt Stanton, and Adrien Treuille. Modular bases for fluid dynamics. *ACM Transaction on Graphics*, 28(3):39:1–39:8, July 2009.

[75] Y. C. Zhou, B. H. Xu, A. B. Yu, and P. Zulli. An experimental and numerical study of the angle of repose of coarse spheres. *Powder technology*, 125:45–54, 2002.