Structure Preserving and Scalable Simulation of Colliding Systems

Breannan Smith

Submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Graduate School of Arts and Sciences

COLUMBIA UNIVERSITY

2018
ABSTRACT

Structure Preserving and Scalable Simulation of Colliding Systems

Breannan Smith

Predictive computational tools to study granular materials are important in fields ranging from the geosciences and civil engineering to computer graphics. The simulation of granular materials, however, presents many challenges. The behavior of a granular medium is fundamentally multi-scale, with pair-wise interactions between discrete granules able to influence the continuum-scale evolution of a bulk material. Computational techniques for studying granular materials must therefore contend with this multi-scale nature.

This research first addresses both the question of how to accurately model interactions between grains and the question of how to achieve multi-scale simulations of granular materials. We propose a novel rigid body contact model and a time integration technique that, for the first time, are able to simultaneously capture five key features of rigid body impact. We further validate this new model and time integration method by reproducing computationally challenging phenomena from granular physics.

We next propose a technique to couple discrete and continuum models of granular materials to one another. This hybrid model reveals a family of possible discretizations suitable for simulation. We derive an explicit integration technique from this framework that is able to capture phenomena previously reserved for discrete treatments, including frictional jamming, while treating bulk regions of the material with a continuum model. To effectively handle the large plastic deformations inherent in the evolution of a granular medium, we further propose a method to dynamically update which regions are treated with a discrete model and which regions are treated with a continuum model. We demonstrate that hybrid simulations of a dynamically evolving granular material are possible and practical, and lay the foundation for further algorithmic development in this space.
Finally, as the tools used in computational science and engineering become progressively more complex, the ability to effectively train students in the field becomes increasingly important. We address the question of how to train students from a computer science background in numerical computation techniques by proposing a new system to automatically vet and identify problems in numerical simulations. This system has been deployed at the undergraduate and graduate level in a course on physical simulation at Columbia University, and has increased both student retention and student satisfaction with the course.
Table of Contents

1 Introduction
   1.1 Discrete Non-Smooth Models: Background and Contributions .............. 2
   1.2 Hybrid Simulation of Granular Materials: Background and Contributions .... 3
   1.3 Automated Verification of Numerical Simulations for Computer Science Instruction 5

2 Structure Preserving Impact Simulation
   2.1 Introduction .................................................. 6
   2.2 Impact ....................................................... 9
   2.3 Simultaneity vs. Propagation .................................. 11
      2.3.1 Simultaneity ........................................... 12
      2.3.2 Propagation ........................................... 13
   2.4 Simultaneity and Propagation ................................... 14
   2.5 Restitution and Inelastic Collapse ................................. 15
   2.6 Friction ..................................................... 18
   2.7 Numerical Implementation ....................................... 19
   2.8 Finite Time Integration ....................................... 21
      2.8.1 From One Instant to Finite Time .......................... 21
      2.8.2 Finite Time Energy Conservation ......................... 21
      2.8.3 Synchronous Time Integration .......................... 22
   2.9 Case Study: 2D Billiard Balls .................................. 23
   2.10 Case Study: 3D Rigid Bodies .................................... 25
      2.10.1 Long-Term Energy Conservation .......................... 25
5 COMS 4167: Physically Based Animation

5.1 Mass-Spring Systems: Theme I Milestone I

5.1.1 Introduction

5.1.2 Academic Honesty Policy

5.1.3 Grading and Lateness

5.1.4 Obtaining and Building the Starter Code

5.1.5 Test Scene Oracle

5.1.6 Virtual Machine Development Environment

5.1.7 XML File Format

5.1.8 Required Features for Milestone I

5.2 Mass-Spring Systems: Theme I Milestone II

5.2.1 Introduction

5.2.2 Clarification on Fixed Degrees of Freedom

5.2.3 New XML Features

5.2.4 Required Features for Milestone II

5.2.5 Scene Scripting

5.3 Mass-Spring Systems: Theme I Milestone III

5.3.1 Introduction

5.3.2 New XML Features

5.3.3 Introduction to Newton’s Method

5.3.4 Introduction to Implicit Euler

5.3.5 Linearized Implicit Euler

5.3.6 Note on ‘Local’ and ‘Global’ Indices

5.3.7 Required Features for Milestone III

5.3.8 Fixed Vertices

5.3.9 Full Implicit Euler (Extra Credit)

5.3.10 Vortex Force (Free Feature for Creative Scene)

5.4 Collisions: Theme II Milestone I
5.4.1 Introduction ........................................... 113
5.4.2 New XML Features .................................... 113
5.4.3 Required Features for Milestone I .................... 114

5.5 Collisions: Theme II Milestone II ......................... 125
5.5.1 Introduction ........................................... 125
5.5.2 New XML Features .................................... 125
5.5.3 Required Features for Milestone II .................... 125

5.6 Collisions: Theme II Milestone III ......................... 137
5.6.1 Introduction ........................................... 137
5.6.2 New XML Features .................................... 137
5.6.3 Required Features for Milestone III .................... 137

5.7 Collisions: Theme II Milestone IV ......................... 147
5.7.1 Introduction ........................................... 147
5.7.2 New XML Features .................................... 147
5.7.3 Required Features for Milestone IV .................... 148

5.8 Rigid Body Simulation: Theme III Milestone I .......... 150
5.8.1 Introduction ........................................... 150
5.8.2 New XML Features .................................... 151
5.8.3 Rigid Body Dynamics .................................. 153
5.8.4 Required Features for Theme III ....................... 161
5.8.5 Food For Thought ...................................... 164
5.8.6 Complications in moving to 3D ......................... 164

5.9 Rigid Body Simulation: Theme III Milestone II .......... 165
5.9.1 Introduction ........................................... 165
5.9.2 New XML Features .................................... 165
5.9.3 The Linear Complimentarity Problem ................... 166
5.9.4 Redundant Constraints ................................ 169
5.9.5 More General Notation ................................ 170
5.9.6 Velocity-Projection for Inelastic Contact ............. 172
5.9.7 Required Features ..................................... 174
List of Figures

2.1 Symmetry preservation requires a box dropped face-first onto a floor to bounce straight back up (left); symmetry breaking can manifest in unexpected ways, as with Gauss-Seidel (right). ......................................................... 8

2.2 Matryoshka dolls: A set of dolls dropped on a table rebounds as a solid object with LCP due to lack of break-away (left doll) but separates with Generalized Reflections (right doll). ................................................................. 11

2.3 Simultaneous vs. propagation perspective: The simultaneous approach of LCP fails to capture the break-away of Newton’s Cradle, while the Gauss-Seidel variant of pairwise propagation fails to capture the symmetry of Bernoulli’s problem. 14
2.4 The danger of propagation with $c_r < 1$: Consider the three-ball Newton’s Cradle example with $c_r = 0$; a single pairwise iteration halves the negative relative velocity at a constraint, implying that pairwise iterations only terminate in the limit with a fixed outgoing velocity. In (a) we plot the post-response velocities of the three balls as a function of $c_r$. Note that a common limiting velocity occurs across a significant range of $c_r$ values, suggesting that within this span, pairwise iterations will require an infinite number of iterations. This is indeed the case as illustrated in (b), where we plot the number of iterations required to reach a collision free state as a function of $c_r$. For $c_r < 0.0717$ the graph oscillates at saturation of maximum iterations – if we performed computations with unlimited precision, an infinite number of iterations would be required. This is known as “inelastic collapse”. Even worse, the range of dangerous $c_r$ values increases with the number of impacting bodies. Consider (c) where, following the analysis of Bernu et al. [Bernu and Mazighi, 1990], we plot the value of $c_r$ at which inelastic collapse occurs as a function of the number of balls in a Newton’s cradle. Observe that the range quickly approaches $c_r = 1$ (the same effect is observed more generally). Finally in (d) we show that in actual numerics, the cost of pairwise propagation (here with pairwise Gauss-Seidel for a box of 10,000 balls) correspondingly blows up as $c_r$ decreases.

2.5 The “cleanest” pool break: A perfectly symmetric, elastic pool break is simulated for 56 balls, starting from symmetric initial conditions. After ten seconds of simulation time GR retains the initial symmetries while Gauss-Seidel orderings all generate unexpected, order-dependent results (the right-hand figure shows one representative Gauss-Seidel simulation).

2.6 Sticky impacts: As in the 1D case, LCP’s inability to resolve breaking contact introduces noticeable sticking artifacts in simulation (bottom). Compare to the breaking splash generated by GR (top).
2.7 Long-term symmetry: Long-term trace of particle trajectories for GR/LCP, Gauss-Seidel and Jacobi. GR, LCP, and Jacobi remain symmetric, while Gauss-Seidel breaks symmetry. Note that LCP does not suffer from ‘sticking’ for this example, and so its solution does not differ from GR’s. Jacobi’s trajectory quickly damps out due to parasitic dissipation.

2.8 Long-term kinetic energy: Over a finite time interval, impact response using Jacobi (solid) yields a random walk type energy behavior, which eventually destabilizes the simulation. GS, GR, and LCP (dashed) exactly conserve kinetic energy, for this system.

2.9 (HAM) and (SYM) with DMV: Here we show the results of three simulations. Top: Eight cubes, initially touching, with symmetric outward velocities constrained to lie in a large cube. There is no gravity in this simulation. Middle: A trefoil knot with threefold rotational symmetry is dropped on a plane under gravity. Bottom: Two bunnies with 180 degree rotational symmetry are released under gravity in a box. Left: For each simulation, we plot traces of points fixed on each body. Right: For each simulation, we plot potential (blue), kinetic (green), and total energy (red). Long term desiderata: Coupled with a time integrator, Jacobi, to the left, respects (SYM) but drops (HAM) while Gauss-Seidel, in the middle, conserves (HAM) but destroys (SYM). Only GR, on the right, satisfies both (HAM) and (SYM) when paired with DMV. To avoid ambiguity in selecting a vertical scale to display the Hamiltonian with, we employ the natural scale defined by the (always positive) kinetic energy.

2.10 Controlled dissipation: As we decrease $c_r$, the overall dissipation decreases in a controlled manner.
2.11 Scaling of our system: Here we show two sequences of configurations from large-scale, 1M-ball simulations in 2D. Impacts are frictionless and elastic (i.e., \( c_r = 1 \)). Top (Drop): Spheres are initialized to a uniform square grid configuration, dropped from rest, under a gravitational potential of magnitude 10. Bottom (Gas): Spheres are initialized to a uniform square grid configuration. Each sphere is assigned an initial, random, unit length velocity. No external forces are exerted on the system.

2.12 Maxwell’s demon demonstrates a dramatic consequence of restitution behavior (\emph{left to right, top to bottom}). Balls placed evenly in two sides of a vibrating box, split by a solid fence, aggregate on one side due to dissipative collisions. The setup here simulates 288 unit mass steel balls with \( c_r = 0.6 \).

2.13 Extended patterns experiment and GR simulation: We simulated the experiments of Melo et al. \cite{Melo1994} across a range of parameters. Top: In the left and middle columns we show side and top view simulation snapshots and note a match with the experimentally obtained square, stripe, and hexagonal patterns in the right column. Bottom: We also reproduce the phase diagram from Bizon et al. \cite{Bizon1998} showing the points that generated both the experimental results and the matching simulations.

3.1 Blurred density: (Left) The reference domain \( \Omega \) of an object with density \( \rho(x) \). Mass density is colored in blue. (Right) A partition of unity of the density mediated by a weight function \( w(x, t) \).

3.2 Hamilton’s principle: (Solid) The physical path traced through configuration space. (Dashed) A small perturbation from the physical path by \( \epsilon \eta \). The physical trajectory will have a stationary action for all local perturbations.

3.3 Hybrid reconciliation zone as a blurred transition: A density partition applied to a partially discrete (left), partially continuum (right) system.

3.4 A granular bed under its own weight: While force chains (black) appear highly non-uniform, homogenization reveals an underlying continuum-like hydrostatic pressure field (high pressure in pink, low pressure in blue).
3.5 Drucker-Prager parameter sweep: We explore the effect of the \( \alpha \) parameter in the Drucker-Prager model by running a series of column collapse simulations with varying values of \( \alpha \). Increasing \( \alpha \) leads to steeper angles of repose in the resulting pile of grains. The ability to tune the repose angle allows us to match the gross behavior of a continuum simulation to a discrete simulation, which enables physically consistent hybrid simulations.

3.6 Initialization of a hybrid simulation: (A) We begin with a collection of discrete grains. (B) We next locate a level set corresponding to a given low density, here denoted as a black line. (C) Across the simulation domain, we compute the distance to the density threshold, here indicated by lines in lighter shades of red as the distance increases. (D) We select a user-tunable distance to the density level-set that will serve as the center of the hybrid ‘reconciliation’ zone. We indicate this critical distance here as a solid black line. (E) We extend the hybrid zone along the distance field by a given width in each direction, indicated here by dotted lines. This hybrid reconciliation zone between the dotted lines defines a zone where the discrete system will be coupled to the continuum system. (F) We delete all discrete grains that fall within the inner boundary of the reconciliation zone. (G) We run the ‘avoid-a-void’ algorithm of Yue et al. [2015] from the outer boundary in to populate the region with material points.

3.7 Granular column: A collapsing column of material simulated with a discrete method (top) and with our hybrid approach (bottom). Observe the nice agreement in the final profile with our hybrid approach and the purely discrete approach.

3.8 Silo discharge: A silo that is discharging grains is simulated with a discrete method (left), and with our hybrid method (right).

3.9 Frictional jamming: A silo discharge simulated with our hybrid algorithm for a large orifice width (left) and for a small orifice width (right). For the large width, grains are able to freely flow from the funnel as the oracle enriches the region near the orifice. For the small width, the granular assembly jams after a tenth of a second, halting all flow from the orifice.
3.10 Penetrometer insertion: We insert a penetrometer into a bed of grains with our hybrid algorithm (first four frames). As the penetrometer enters the bed, our hybrid oracle identifies the region around the tip as requiring a discrete treatment and enriches the simulation domain in this area. As the simulation progresses, the continuum region eventually experiences a topology change and splits in two. Examining an overlay of a purely discrete simulation on top of a hybrid simulation, we find the resulting profiles to be in almost perfect agreement (rightmost frame).  

3.11 Spinning drum: We rotate a drum filled halfway with granules with a discrete element method (top row) and with our hybrid algorithm (bottom row). As the system evolves, observe that the shape of the free surface obtained with our hybrid method agrees with that of the purely discrete method.  

3.12 3D column collapse: A cutaway rendering of a column collapse computed using our preliminary 3D hybrid algorithm. Material points are hidden to reveal the region treated with a continuum.  

4.1 Drifting trajectories under floating point arithmetic. Bottom: Small, $\epsilon$-sized differences in two implementations’ output can lead to differences greater than $\epsilon$ over multiple iterations of the algorithm. Top: Reseeding one implementation’s state with another implementation’s state at the start of each time-step prevents global drift due to acceptable $\epsilon$-sized differences in output, allowing a direct comparison of the implementations’ output.  

4.2 Simulation sensitivity. An $\epsilon$-sized delta in state in a single time-step can lead to a massive global divergence in a system’s trajectory. As a tangible example, we simulate a circular ball colliding against a fixed box. On the left we impose an $\epsilon$-sized clockwise rotation of the box, while on the right we impose an $\epsilon$-sized counterclockwise rotation of the box. While these changes lead to small changes in a single time-step, they produce large differences in the global trajectory, complicating a direct comparison of the simulations’ output.
4.3 Autograder in practice: A student implements a course assignment, and runs an instructor-provided simulation description through their simulation code. The student can then load this simulation output and the simulation description into the autograder, which compares their simulation output against a known, vetted implementation of the algorithm under study. The autograder provides feedback to help the student debug the assignment if their implementation is incorrect. Finally, when the assignment is due, the student’s implementation is tested with a large corpus of input simulation descriptions using the autograder, and their final grade is determined by the number of simulation descriptions that are successfully performed by the student’s code.

5.1 Geometry of particle interactions: The separation distance and the normalized separation vector used to compute pairwise spring and gravitational forces between particles.

5.2 Effect of integrator choice: Comparison of orbits computed with explicit Euler (left) and forward-backward Euler (right).

5.3 Newton’s method: The geometry of Newton’s method illustrated in one dimension. Linearizing the function around $x_i$ and intersecting the line with the horizontal axis yields the next iterate $x_{i+1}$.

5.4 Notation for implicit integration: A summary of the notation we use to derive implicit Euler.

5.5 Particle vs. edge collisions: Left: Checking whether or not a particle and an edge are colliding. Several vectors between the particle and the edge are drawn in blue; the shortest one $n$ is drawn in red. Since $n$ has length greater than the sum of the two objects’ radii, the objects are not overlapping and so are not colliding. Right: Two edges cannot collide without a particle-edge collision also occurring (particle drawn in red).

5.6 Particle collision normals: Left: A particle–particle collision. Middle: A particle–edge collision. Right: A particle–half-plane collision. In each case, the shortest vector between the objects, $n$, has been drawn in red.
5.7 Limitations of discrete time collision detection: Simply checking for collisions at the end of the time-step misses some collisions, such as this situation where a particle with high velocity tunnels through an edge. ........................................ 126

5.8 Systems of inequalities: Solving the system of inequalities \( f(t) > 0, \ g(t) > 0 \) for two example polynomials. Left: Plots of the two polynomials. Right: The intervals of \( t \) on which \( f \) and \( g \) are positive, and the intersection (bottom, in red) of the two sets of intervals. Any point inside a red interval is a solution to the system of inequalities. ........................................ 128

5.9 Effect of an approximate \( \alpha \): A fast-moving particle collides with an edge at a glancing angle. If we assume \( \alpha \) is constant, the dot product between \( \mathbf{n}(t) \) (red arrow) and \( \Delta \mathbf{x} \) (black arrow) is negative at the time of collision, and we (incorrectly) conclude that the particle and edge are separating at that time. ........................................ 133

5.10 Penalty methods: A conceptual illustration of the penalty method. A spring-like penalty force repels \( O_1 \) and \( O_2 \), since they are interpenetrating, and \( O_1 \) and \( O_3 \), since they are sufficiently close to each other (less than distance \( T \) apart, where \( T \) is the penalty method thickness). No force is exerted between \( O_2 \) and \( O_3 \) because they are far apart. ........................................ 134

5.11 Iterative collision response: An example showing that a single round of collision detection and response can be insufficient. Initially, particles 1 and 2 are detected as colliding (left); applying impulses resolves this collision, but introduces a new collision between particles 2 and 3 (middle left). A second iteration of collision detection and response (middle right) is necessary to produce a collision-free simulation at the end of the time-step (right). ........................................ 138

5.12 Geometric collision response: An illustration of the geometric collision response algorithm. Given a set of colliding vertices (left), we convert them into a rigid body (middle left), step the rigid body forward to the end of the time-step (middle right), then convert the rigid body back into individual particles (right). The resulting positions are guaranteed to be collision-free. ........................................ 140
5.13 Impact zones: A number of collisions are detected during a time-step (left). Based on these detected collisions, the particles in the simulation involved in collisions are separated into impact zones (right). Each impact zone is a different color.

5.14 Iterative geometric collision handling: Some collisions are detected, and impact zones constructed (left). Stepping the simulation forward causes new collisions (middle left), so the impact zones grow (middle). This process repeats a second time (middle right and right).

5.15 The hull of a rigid body: We describe the shape of a rigid body by a sequence of vertices. Note the clockwise specification of vertices.

5.16 Rigid body transformations: The body space and world space frames.

\[ r_i' = R(t)r_i^0 \]
\[ r_i = R(t)r_i^0 + x(t). \]

5.17 Angular velocity: A rigid body rotating about its center of mass and axis \( \omega \).

5.18 Wind force notation: Subdivision of an edge for computing the wind force. Here the number of sample points is four.

5.19 Breaking contacts in a rigid body simulation: Active contacts are denoted by small stars, breaking contacts by small circles. Naively resolving collisions in an iterative fashion can produce an incorrect solution.

5.20 Breaking contact example: Block resting on table with center of mass over the edge.

5.21 Static indeterminacy: A table under gravity in two dimensions with three contacts. Observe that multiple sets of impulses give the same solution - we could apply only the large impulse to the center leg, or the two smaller impulses to the outer legs.

5.22 Velocity projection: On the left, unit-mass particles moving in configuration space constrained to lie above the horizontal. On the right, the same particles in velocity space and the constraint impulses. Post-impulse velocities are denoted by stars. Notice that the post-impulse velocities are given by the closest point in the admissible region.
List of Tables

2.1 Multi-impact feature chart: Physics demands a great deal from a multi-impact solution, and previous models punt on one goal or another. .................. 7

2.2 Timings: Performance statistics for balls constrained to a drum. For each simulation we report the wall-clock time (seconds) spent in unconstrained integration, collision detection, and impact response per time-step. We report the min, max, and average time across all time-steps. We also report the number of iterations spent in each GR solve, as well as the overall active set size. The system was integrated with a fixed time-step of 0.01s and employed the nonlinear optimization package Ipopt to solve the impact QP. Timings were recorded with a single thread on a 3.33GHz Intel Core i7-975. .................... 28

2.3 Hexagonal extended pattern statistics: Performance for decreasing time-step sizes. Timings (reported above in seconds) were recorded with a single thread on a 2.67GHz Intel Xeon 5650 for a system of 360,000 DoFs. .................. 30
Acknowledgments

Thank you to my thesis advisor, Eitan Grinspun. Without his support and guidance, none of this work would have come to fruition.

Thank you to my research collaborators during my doctoral studies: Jean-Marie Aubry, Christopher Batty, Maytee Chantharayukhonthorn, Yichen Chen, David Harmon, Ken Kamrin, Danny Kaufman, Rasmus Tamstorf, Etienne Vouga, Yonghao Yue, and Changxi Zheng. Your passion for science and the pursuit of knowledge is a never ending inspiration. Thank you to everyone in the Columbia Computer Graphics Group, one could not ask for a more energizing and dynamic group of colleagues. Thank you to Layne McNish for putting up with far too many late nights, early mornings, and weekend sprints during looming SIGGRAPH deadlines. Finally, thank you to all of my friends and family, your support was critical to the completion of this work.
Granular materials, as a ubiquitous component of our natural world, are an important topic of study in fields ranging from the geosciences and film visual effects to robotics. Granular materials trail only water as the most commonly handled industrial material, and a predictive understanding of granular materials is key to the industrial scale processing of construction materials, pharmaceuticals, food, and sand. Recently, the failure to understand the properties of the underlying soil has caused a $350,000,000.00 residential tower in San Francisco to slowly sink into the ground, with remediation estimated to cost over $100,000,000.00. Despite their importance, formulating predictive models of granular materials presents many challenges. Under different loading conditions, granular materials can exhibit behaviors characteristic of an elastic solid (forming static piles), of a liquid (flowing plastically and sustaining permanent shape change), and of a gas (individual grains in free flight ballistic motion). Compounding the difficulties stemming from this multi-phase nature, granular materials are fundamentally multi-scale, with individual grains typically too large to admit a continuum assumption. This thesis addresses the simulation of granular materials from two perspectives. We begin by proposing a novel model for collisions between grains that, for the first time, provably respects five critical properties from the underlying physics. We then propose a method to couple a discrete model of a granular medium to a continuum model, leading to a hybrid simulation technique that inherits benefits from both approaches. Finally, we propose a new methodology to more effectively train computer science students in the tools of computational...
CHAPTER 1. INTRODUCTION

1.1 Discrete Non-Smooth Models: Background and Contributions

Modeling a granular medium as a collection of discrete bodies governed by grain vs. grain interactions is a natural approach, with roots in the work of Cundall and Strack \cite{Cundall1979}. While Cundall and Strack model these interactions as ‘soft’, penalty-based forces, an alternate approach models the interactions through constraints, leading to so-called ‘non-smooth’ methods. Non-smooth algorithms and models for the simulation of rigid multi-body systems dominated by contact, impact, and frictional forces have undergone rapid advancement and adoption since their conception \cite{Brogliato1999, Jean1999b, Stewart2000a, Bender2014}. Non-smooth methods promise to remedy many of the shortcomings of penalty approaches, namely their difficult to tune parameters and strict time step restrictions. In concert with the development of non-smooth contact models, geometric, numerical integrators have proven to be powerful tools, providing time stepping methods that conserve core structures from the underlying physical systems even in the face of coarse temporal and spatial discretizations \cite{Hairer2002}. Attempts to marry the benefits of structure preserving integrators to smooth contact models have proven fruitful \cite{Harmon2008, Vouga2011}, but the non-smooth problem remains difficult. As discussed by Kaufman and Pai \cite{Kaufman2012}, attempts to directly substitute non-smooth forces into a variational framework or to interleave non-smooth collision events with variable length steps of a smooth variational integrator do not maintain the structure preserving properties of the underlying variational integrators. We have successfully developed a model and integrator that captures and preserves key features from the underlying physics even when discretized. This has resulted in three key contributions:

The First Non-Smooth Collision Model to Satisfy Five Key Desiderata. We identify five important properties of simultaneous collisions between rigid bodies: simultaneous collisions should conserve momentum, elastic collisions should conserve kinetic energy while collisions with a coefficient of restitution should reduce kinetic energy, collisions should should
CHAPTER 1. INTRODUCTION

push bodies apart but not pull them together, collisions should preserve spatial symmetries, and collisions should propagate shock waves. Surprisingly, prior to the introduction of our collision model, no non-smooth simultaneous collision model was able to respect all five properties at once.

A Solution to Inelastic Collapse. Non-smooth simultaneous collision models that propagate shocks under a coefficient of restitution suffer from a problem known as inelastic collapse [Baraff, 1989a; McNamara and Young, 1994], leading to the non-termination of this propagation. Our model is able to capture shock propagation effects with a coefficient of restitution, but is free from the problem of inelastic collapse.

A Structure Preserving Discrete Time Integrator. With a collision model in place, we develop a time integration technique that continues to respect the five established desiderata under discrete time-stepping.

The net deliverable from these developments is a technique for simulating granular materials that only contains model parameters. As a consequence, we are able to directly use parameters reported in physical experiments, with the resulting simulations in agreement with the experimentally reported results.

1.2 Hybrid Simulation of Granular Materials: Background and Contributions

Existing efforts to model granular materials often focus on special cases. Continuum models from solid mechanics, while well suited to model the deformation of dense soils, are not able to capture all behaviors of general import: small scale rapid collisions near free surfaces allow individual grains to separate from the bulk material, the finite size nature of individual grains can lead to frictional jamming in narrow chutes or during drainage scenarios, and grain-scale shear banding can mediate larger scale deformations.

Crucially, in each of these phenomena grain-scale dynamics shape the evolution of the continuum-scale bulk. On the other hand, discrete particle models specifically treat grain-scale interactions, and in practice are often used as a ground-truth validation for continuum models [Rycroft et al., 2009]. Discrete particle models are computationally intractable when scaled to
large domains, however, which severely limits their practical applicability. We are seemingly faced with a trade-off: simulate a granular system with a continuum model but lose fidelity and generality, or simulate a granular system with a discrete particle method but limit the scope to a small sub-domain.

To resolve this trade-off, we propose a method that treats all phases of granular materials, across scales, by unifying continuum and discrete treatments in a hybrid simulation algorithm. Our hybrid treatment is able to capture grain-scale phenomena while at the same time retaining the superior scaling of a continuum model. In formulating a hybrid method to simulate granular materials, we introduce three fundamental contributions: a framework for coupling a discrete particle model to a continuum model, an explicit integration method derived from this framework that couples a penalty based discrete model to a material point based continuum simulation, and an adaptive homogenization and coarsening technique to dynamically covert between continuum and discrete representations as a simulation evolves.

**A Framework for Hybrid Granular Simulations.** Lagrangian mechanics provides a natural, variational setting for reasoning about and deriving equations of motion in the presence of constraints. By dividing a system into two systems whose masses form a partition of unity of the original system’s mass and by imposing a constraint on the velocities of the systems, we derive an Arlequin-type approach for coupling two distinct models. When discretized, we obtain a general framework for building hybrid integration techniques for granular materials.

**An Efficient, Explicit Integrator for Hybrid Granular Simulations.** From this general framework, we derive a particular explicit integration technique with favorable properties. By coupling a discrete particle simulation to a material point method simulation, we are able to exploit the specific structure of the material point method and avoid the need for a linear solve. This results in a compact method for time-stepping the coupled systems that is trivially parallelized.

**Homogenization and Coarsening of Hybrid Granular Simulations.** Flowing granular systems can undergo large plastic deformations and can experience topology changes. As a result, portions of the material that were originally suited for a continuum treatment might require discrete degrees of freedom. Similarly, portions of the material that were initially represented with a discrete treatment might be safely treated with a continuum as the simulation evolves. 
evolves. We present a technique to identify which regions should be treated with a discrete method, and a method to convert between discrete and continuum representations of the material. This technique allows us to perform robust simulations even in the face of massive plastic shape changes.

1.3 Automated Verification of Numerical Simulations for Computer Science Instruction

Building on this research will require future generations of computer scientists who can effectively use the tools of computational science and physically based animation. A major drawback in teaching these tools is that physically based animation draws from a different scientific and mathematical background than what is typically taught to undergraduate computer science students. Compounding this issue, our course in physically based animation is often these students' first exposure to writing and debugging numerical heavy code. To ease their introduction to the field, we propose a methodology for teaching this course material in a manner that more closely mirrors the structure that computer science students are accustomed to.

A Framework for Hybrid Granular Simulations. The key component to our new course structure is a novel system for automatically verifying the correctness of each student’s numerical code. Our so-called auto-grader is able to vet the correctness of numerical simulations implemented by students, while at the same time providing instant, visualizable information to help locate and eliminate errors. This is accomplished through the use an ‘oracle’, which runs a student’s simulation in tandem with a known, verified simulation code in a manner that is robust to numerical drift between the implementations.
Chapter 2

Structure Preserving Impact Simulation

2.1 Introduction

Modeling the dynamics of sustained contact (resting, sliding) and of instantaneous impact (transient collisions, bouncing) is a fundamental thread of research in graphical simulation [Hahn, 1988; Baraff, 1989a; Mirtich and Canny, 1995; Witkin and Baraff, 2005]. We focus on the open problem of modeling multi-impact, where either as a result of time-discretization or by consequence of formulation, we are asked to simultaneously resolve multiple collisions occurring at an instant. Models (e.g., sustained contact) and methods (e.g., penalty) that require finite time intervals to act are not considered here.

Consider arrangements of three balls at the instant they all collide:

<table>
<thead>
<tr>
<th></th>
<th>Before</th>
<th>Pre-impact</th>
<th>Post-impact</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bernoulli’s Problem</td>
<td><img src="image1" alt="Diagram" /></td>
<td><img src="image2" alt="Diagram" /></td>
<td><img src="image3" alt="Diagram" /></td>
<td><img src="image4" alt="Diagram" /></td>
</tr>
<tr>
<td>Newton’s Cradle</td>
<td><img src="image5" alt="Diagram" /></td>
<td><img src="image6" alt="Diagram" /></td>
<td><img src="image7" alt="Diagram" /></td>
<td><img src="image8" alt="Diagram" /></td>
</tr>
</tbody>
</table>
In each case, leaving pre-impact velocities unchanged leads to penetration. These velocities must therefore be altered via instantaneous impulses to avoid penetration, i.e., to become feasible. What does physics tell us about the requisite impulses and the attendant post-impact velocities?

(BRK) **Break away.** Bodies that were previously in contact may break away from each other as a result of impact. This might occur as an immediate consequence of the impact, as in Bernoulli’s Problem, or it may be the result of shock propagation—a sequence of ordered events occurring at an instant—as in Newton’s Cradle.

(SYM) **Symmetry preserved.** Spatial symmetries (e.g., about a reflection line) that exist in pre-impact configurations should also exist in post-impact configurations. After all, in an ideal system, what factor breaks such a symmetry [Bernoulli, 1742]? As depicted above, both Bernoulli’s Problem and Newton’s Cradle are symmetric about the horizontal bisector.

(KIN) **Energy bounded.** Elastic impact ($c_r = 1$) conserves kinetic energy. Inelastic impact under a coefficient of restitution ($0 \leq c_r < 1$) reduces kinetic energy. Barring the esoteric case $c_r > 1$, kinetic energy does not increase.

(MOM) **Momentum conserved.** Because impacts are internal to the (closed system of bodies, total momentum is conserved.

(ONE) **One-sided impulses.** Impulses may push bodies apart but may not pull them together, the so-called “no-velcro” condition.

These five physical desiderata define our notion of a correct model for instantaneous impact. Given these goals, how well do existing families of models perform?

We summarize the answer in Table 2.1. Unfortunately, existing models, taxonomized by their use of a linear complementary problem (LCP), Gauss-Seidel, or Jacobi structure (§2.3), fail to satisfy either break-away, symmetry preservation, or kinetic energy conservation.

**Contributions** We propose a generalized reflections multi-impact operator that satisfies all five desiderata. In its simplest interpretation the approach we will present amounts to a care-
CHAPTER 2. STRUCTURE PRESERVING IMPACT SIMULATION

<table>
<thead>
<tr>
<th>Model</th>
<th>(BRK)</th>
<th>(SYM)</th>
<th>(KIN)</th>
<th>(MOM)</th>
<th>(ONE)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Our model</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
</tr>
<tr>
<td>LCP</td>
<td>☒</td>
<td>☐</td>
<td>☐</td>
<td>☐</td>
<td>☑</td>
</tr>
<tr>
<td>Gauss-Seidel</td>
<td>☐</td>
<td>☒</td>
<td>☐</td>
<td>☐</td>
<td>☑</td>
</tr>
<tr>
<td>Jacobi</td>
<td>☐</td>
<td>☐</td>
<td>☒</td>
<td>☐</td>
<td>☑</td>
</tr>
</tbody>
</table>

Table 2.1: Multi-impact feature chart: Physics demands a great deal from a multi-impact solution, and previous models punt on one goal or another.

ful combination of the LCP and Gauss-Seidel formulations. And while forming hybrids of two methods is often a recipe for aggregating flaws, in this case, the resulting *generalized reflection operator* provably fulfills all desiderata.

**Practical implications** Beyond the satisfaction of capturing the inherent beauty of the physical laws, the fulfillment of the desiderata offers important practical benefits:

*Breaking contact* is an experimentally validated and expected behavior in stiff impact, as in the propagation of a shock in Newton’s Cradle: impact from the leftmost ball propagates, traversing the chain, until the rightmost ball breaks away. On high-speed film it is observed that the propagation is *effectively instantaneous*, in that it traverses the domain at time scales separated by orders of magnitude from the gross dynamics.

![Figure 2.1](image.png)

Figure 2.1: Symmetry preservation requires a box dropped face-first onto a floor to bounce straight back up (left); symmetry breaking can manifest in unexpected ways, as with Gauss-Seidel (right).

Shock waves have been observed to propagate through a contacting medium in both a time-ordered [Hascoët et al., 1999; Pudasaini and Kröner, 2008] and also in a *symmetry-
Artificial symmetry breaking can yield unexpected results. Consider the simple canonical case of a rigid box dropped face-first onto a horizontal floor. At impact, standard box/box collision routines detect all four face vertices as colliding. Constraints are then generated by assigning these corner vertices the floor normal. Since the projection of the box’s pre-impact velocity (i.e., its rigid-body twist) onto all four constraint normals is identical, the box’s post-impact velocity should retain this symmetry and thus bounce straight back up (Fig. 4.1 left). Post-impact velocities that violate this symmetry, but otherwise satisfy all four remaining desiderata, contain non-zero angular velocity components that cause the box to incorrectly fly away along an oblique trajectory (Fig. 4.1 right). More generally, all symmetries between pre-impact velocities and constraints should be preserved by post-impact velocities.

**Bounded energy**, in both the conservative ($c_r = 1$) and dissipative ($c_r < 1$) regimes, leads to stable simulations. Energy behavior (whether conservative or dissipative) that remains independent of evaluation order and permutation of degrees of freedom (DoFs) provides more consistent simulations. Here we develop a model that guarantees bounded energy for the instantaneous resolution of simultaneous impacts.

**Overview** Our story begins by considering a single instant in time. We first explore the uniqueness of solutions in the case of multiple impacts (\S\ 2.2). Through a careful analysis of existing models (\S\ 2.3) we then arrive at our new generalized reflections operator (\S\ 2.4). In the presence of restitution, inelastic collapse challenges termination of this and other impact models. However, we show how such collapse can be entirely avoided with the application of a simple energetic restitution model (\S\ 2.5). To conclude our instantaneous story we present a compatible friction formulation (\S\ 2.6) and a scalable numerical implementation (\S\ 2.7).

## 2.2 Impact

Impacts and contact occur whenever we impose one-sided constraints between objects or DoFs. A constraint is expressed by an inequality $g_i(q) \geq 0$, where the constraint function $g_i : Q \rightarrow \mathbb{R}$ maps each configuration $q \in Q$ to a non-negative number iff $q$ is admissible. Trajectories, $q(t) \in Q$, are then required to be non-negative, $g_i(q(t)) \geq 0$, for all time $t \in [0, T]$. 

Our story takes place at instants where objects touch, i.e., \( g_i(q) = 0 \). Differentiating the constraint with respect to time then gives

\[
\nabla g_i(q)^T \dot{q} \geq 0.
\]

We cannot sustain a velocity opposing the normal \( \nabla g_i(q) \).

**Contact** treats the case \( \nabla g_i(q)^T \dot{q} \geq 0 \) when \( g_i(q) = 0 \).

**Impact** treats the case \( \nabla g_i(q)^T \dot{q} < 0 \) when \( g_i(q) = 0 \). Here the pre-impact velocity \( \dot{q}^- \) opposes a constraint normal, necessitating an *impulsive* change to yield a post-impact velocity \( \dot{q}^+ \) satisfying (2.1).

**Isolated impact(s)** At an instant when exactly one constraint experiences impact, \( \dot{q}^+ \) is uniquely determined from conservation of momentum by the (\( c_r \) scaled) reflection

\[
\nabla g(q)^T \dot{q}^+ = -c_r \nabla g(q)^T \dot{q}^-.
\]  

Two cases have special properties: *elastic* impact (\( c_r = 1 \)) conserves energy; *purely inelastic* impact (\( c_r = 0 \)) dissipates more energy than any other momentum-respecting response.

At an instant when *two* constraints experience impact, we might get lucky with an easy case: if the normals \( \nabla g_1(q) \) and \( \nabla g_2(q) \) are orthogonal then each constraint is safely isolated as above. Two independent collisions across the room from each other, for example, possess orthogonal normals. This lucky strike generalizes to *n* simultaneous impacts, when all *n* normals are mutually orthogonal.

**Multi-impact** A more interesting case occurs at an instant where *n* constraints experience impact, with general (not necessarily orthogonal) constraint normals \( \nabla g_1(q), \ldots, \nabla g_n(q) \). This is the typical situation when multiple bodies collide at once, when a body collides against another with multiple points of contact, or when a particle collides against a kink of an enclosing boundary.

In the elastic case the core properties of elastic impact appear to delimit, but not uniquely pin down, the impulse. This issue is raised in multiple works [Moreau, 1988a](#), [Ivanov, 1995](#), [Brogliato, 1999](#), [Chatterjee and Ruina, 1998](#), [Glocker, 2004](#) that analyze the time-continuous setting. In essence, we understand how a particle bounces off of a wall, but not
how it bounces off of an arbitrarily-shaped corner. Since core conservative properties do not uniquely prescribe impact at a kink, we seek a principled way of choosing one such canonical outbound trajectory out of the myriad options.

Although the resolution of multi-impact is tricky, it would be dangerous to treat multi-impact as a degenerate case, since in practice the resolution of multiple impacts is the typical case. For example, (a) when synchronous time-stepping methods advance, all constraints violated en route are treated as simultaneous collisions \cite{Bridson2002}; (b) asynchronous time of impact (TOI) methods are used to compute the motion of huge, multi-body systems \cite{Lubachevsky1991}, here the probability of multiple simultaneous impacts increases with problem size; (c) for dissipative physical systems, such as those with $c_r < 1$, the average time between collisions gradually decreases, consequently the probability of multi-impact increases (until ultimately a stable sustained contact forms).

\section{Simultaneity vs. Propagation}

Existing multi-impact models fall into two categories \cite{Baraff1989a}: they focus on either simultaneity or propagation, as exemplified by Bernoulli’s and Newton’s problems, respectively.

\subsection{Simultaneity}

The Linear Complementarity Program (LCP) approach to multi-impact begins by determining the active constraint set $A(q) = \{i : g_i(q) = 0\}$, and then requires the impact-level Signorini-Fischera condition \cite{Moreau1983b, Baraff1989a, Stewart2000a}

$$
0 \leq \lambda \perp G_A^T \dot{q}^+ \geq -c_r G_A^T \dot{q}^-,
$$

where $G_A$ is the matrix with columns $\{\nabla g_i(q) : i \in A\}$, $\lambda \in \mathbb{R}^{|A|}$ is the vector of impulse coefficients, and $x \perp y$ is the complementarity condition $x_i y_i = 0$, $\forall i$. Because $G_A$ is not generally full-rank, the $\lambda$ satisfying the Signorini-Fischera condition is not necessarily unique; nevertheless, when scaled by the inverse mass matrix $M^{-1}$, this impulse leads to a unique post-impact velocity

$$
\dot{q}^+ = \dot{q}^- + M^{-1} G_A \lambda.
$$
Benefits and failures of LCP

As the LCP considers only spanning spaces \((\text{span } G_A)\), ignoring the choice of basis vectors \((\nabla g_i)\), it is geometric in the sense of being basis- (or coordinate) independent. Concretely, it (a) produces a result independent of DoF or sampling permutations and (b) preserves symmetry by construction.

Furthermore, for purely inelastic impact \((c_r = 0)\), the LCP formulation produces the unique solution that maximally dissipates normal velocities \cite{Moreau1983}, satisfying all desiderata.

Indeed, LCPs satisfy one-sidedness (ONE) for any \(c_r\), by construction. This, then, serves to highlight the distinction between one-sidedness (ONE) and break-away (BRK). For the elastic case \((c_r = 1)\), LCPs do not satisfy (BRK) \cite{Chatterjee1998, Glocker2004}: the LCP solution of the elastic Newton example exhibits sticking where we expect break-away (see Fig. 2.3). Intuitively, the LCP solution reflects each relative velocity; when the relative velocity is zero (sustained contact), the reflected relative velocity remains zero, so that LCP has pinned the constraint, instead of allowing breaking contact. Put precisely, LCPs “stick” by producing zero (rather than positive) post-impact relative velocities, \(\nabla g_i(q)^T \dot{q}^+ = 0\), at contact points where corrective impulses, \(\lambda_i > 0\), have been applied. As a consequence, LCP solutions
do not capture shock-propagation effects.

Thus, while LCPs might be the ideal solution for purely inelastic multi-impact, as given below in Alg. 1, LCPs do not and can not correctly treat multi-impact in general.

**Algorithm 1 Inelastic_Impact(q, p, A)**

1: $G \leftarrow G_A(q)$
2: $\lambda \leftarrow \text{argmin}_y \left( \frac{1}{2}(Gy + p)^T M^{-1}(Gy + p) : y \geq 0 \right)$
3: return $\lambda$

LCP’s behavior for our two model problems summarizes its strengths and weaknesses as a method for elastic impact. LCP produces the correct behavior for Bernoulli’s problem because it preserves symmetry. However, LCP produces incorrect sticking for Newton’s Cradle (see Fig. 2.3).

### 2.3.2 Propagation

**Pairwise propagation models** leverage the well-posed behavior and computational ease of resolving a single-point impact. Dating back to the foundations of impact mechanics [Maclaurin, 1742; D’Alembert, 1743], these methods sequentially resolve each collision in isolation. Because each collision is treated separately, the communication between collisions occurs explicitly when a previously-treated collision is revisited; in general, propagation models iterate multiple times over the full set of active collisions.

Some variants use random ordering for the sequence [Ivanov, 1995], others invoke physical [Chatterjee and Ruina, 1998] or geometric [Johnson, 1976; Ivanov, 1995] considerations. Two popular variants [Bridson et al., 2002; Guendelman et al., 2003] mimic the styles of the simplest iterative linear solvers: the *Jacobi* variant first computes every pairwise impulse, and then applies them all [Maclaurin, 1742], whereas the *Gauss-Seidel* variant computes and applies each impulse in turn [D’Alembert, 1743; Johnson, 1976; Ivanov, 1995; Chatterjee and Ruina, 1998]. All of these variants are not to be confused with the similar-sounding names of iterative splitting methods used to solve LCPs [Cottle et al., 1992; Erleben, 2007a].

---

The inelastic LCP given by (2.3) with $c_r = 0$ is the KKT optimality condition [Boyd and Vandenberghe, 2004] for the equivalent minimization employed in Alg. 1, line 2. We denote the system’s momentum as $p = Mq$. 

---
Benefits and failures of pairwise propagation  Pairwise propagation models enable both breaking contact and shock-propagation. However, all such methods either violate energy conservation, e.g., Jacobi (see Sec. 2.9), leading to large instabilities or excess dissipation, or destroy symmetry, e.g., Gauss-Seidel (see Fig. 4.1 right), and thus produce non-physical and inconsistent trajectories (see also Figs. 2.3, 2.7, and 2.5).

Figure 2.3: Simultaneous vs. propagation perspective: The simultaneous approach of LCP fails to capture the break-away of Newton’s Cradle, while the Gauss-Seidel variant of pairwise propagation fails to capture the symmetry of Bernoulli’s problem.

Recap  Thus far, we have observed that the standard simultaneous LCP impact model is only well-behaved for $c_r = 0$, but otherwise is prone to sticking. On the other hand, while pairwise propagation models recover break-away behavior, they violate either energy conservation or symmetry preservation.

2.4 Simultaneity and Propagation

Generalized reflection  We begin our development with a simple, yet critical, observation:

Lemma. In the special case where the approaching velocity opposes all constraint normal directions, the LCP model generates a unique solution that is always free of both sticking artifacts and feasibility violations, for any $0 \leq c_r \leq 1$.

Proof. The LCP in (2.3) generates a unique post-impact velocity for all feasible problems [Cottle].
et al., 1992. By assumption, all normal velocities are initially negative. Then, by the Signorini-Fischera condition,

$$G^T_A q^+ \geq -c_r G^T_A q^- > 0,$$

for all $c_r > 0$.

In particular, for this special case, the LCP solution is a generalized reflection, in the sense that the component of the velocity in the subspace spanned by all pushing normal directions (i.e., directions along which an impulse is applied) is negated (and scaled by $c_r$), while the component in tangential directions remains untouched. The solution also inherits the energy and symmetry preservation properties inherent to the LCP formulation.

**Generalized reflections impact operator** During multi-impact, the incoming velocity will generally oppose some, but not all normal directions. Let these opposing normals define a violator subset of the active set of constraints, $V \subset A$. If we temporarily ignore the remaining active constraints and apply LCP just to the violator subset, our special-case generalized reflection, described above, yields the unique impulse that recovers all core properties. This feasible “post-impact” velocity no longer opposes the violator normals.

However, having ignored the remaining active constraints, this velocity may not be feasible with respect to the full set of original constraints. Thus, again, we gather the set of opposing normals, apply the generalized reflection, and repeat (Alg. 2).

This resulting Generalized Reflections (GR) impact operator consisting of back-to-back special-case impulses, directly corresponds to the view of a shock wave as an advancing front of constraint violations. A shock wave can be viewed as an instant in time—positions are fixed—during which a moving front of velocity modifications sweeps over the material. In particular, feasibility is satisfied everywhere except on the moving front, where constraint-restoring impulses induce new violations ahead of the front and thus advance the front forwards.

### 2.5 Restitution and Inelastic Collapse

Pairwise propagation methods are well known to suffer from poor convergence whenever $c_r < 1$ is applied; in the extreme case, they cannot converge in finite iterations due to inelastic
Figure 2.4: The danger of propagation with $c_r < 1$: Consider the three-ball Newton’s Cradle example with $c_r = 0$; a single pairwise iteration halves the negative relative velocity at a constraint, implying that pairwise iterations only terminate in the limit with a fixed outgoing velocity. In (a) we plot the post-response velocities of the three balls as a function of $c_r$. Note that a common limiting velocity occurs across a significant range of $c_r$ values, suggesting that within this span, pairwise iterations will require an infinite number of iterations. This is indeed the case as illustrated in (b), where we plot the number of iterations required to reach a collision free state as a function of $c_r$. For $c_r < 0.0717$ the graph oscillates at saturation of maximum iterations – if we performed computations with unlimited precision, an infinite number of iterations would be required. This is known as “inelastic collapse”. Even worse, the range of dangerous $c_r$ values increases with the number of impacting bodies. Consider (c) where, following the analysis of Bernu et al. [Bernu and Mazighi, 1990], we plot the value of $c_r$ at which inelastic collapse occurs as a function of the number of balls in a Newton’s cradle. Observe that the range quickly approaches $c_r = 1$ (the same effect is observed more generally). Finally in (d) we show that in actual numerics, the cost of pairwise propagation (here with pairwise Gauss-Seidel for a box of 10,000 balls) correspondingly blows up as $c_r$ decreases.
CHAPTER 2. STRUCTURE PRESERVING IMPACT SIMULATION

Algorithm 2 Generalized Reflection(q, p, A, c_r)
1: \( \epsilon \leftarrow 1 + c_r \)
2: \( \lambda \leftarrow 0 \)
3: \( \tilde{p} \leftarrow p \)
4: while true do
5: \( \forall \leftarrow \emptyset \)
6: for \( k \) in \( A \) do
7: if \( \nabla g_k(q)TM^{-1}\tilde{p} < 0 \) then
8: \( \forall \leftarrow k \)
9: end if
10: end for
11: if \( \forall \neq \emptyset \) then
12: \( G \leftarrow G_{\forall}(q) \)
13: \( \tilde{\lambda} \leftarrow \text{argmin}_y \left( \frac{1}{2}(Gy + \epsilon\tilde{p})TM^{-1}(Gy + \epsilon\tilde{p}) : y \geq 0 \right) \)
14: \( \tilde{p} \leftarrow \tilde{p} + G\tilde{\lambda} \)
15: \( \lambda \leftarrow \lambda + \tilde{\lambda} \)
16: else
17: return \( \lambda \)
18: end if
19: end while

collapse [Baraff, 1989a; McNamara and Young, 1994]. A simple exercise is to observe that for Newton’s Cradle with \( c_r = 0 \), each iteration halves the negative relative velocity, thus by Zeno’s Paradox (“Achilles and the tortoise”) finite iterations cannot reach a feasible velocity.

More generally, inelastic collapse extends well above the fully inelastic case of \( c_r = 0 \) (see Fig. 2.4 (a) and (b)). Indeed, the range of \( c_r \) for which inelastic collapse can occur increases as the size of an impacting system grows (see Fig. 2.4 (c)), and quickly approaches unity as the number of colliding bodies become sufficiently large [Bernu and Mazighi, 1990][McNamara and Young, 1994]. Thus inelastic collapse is effectively unavoidable for the large-scale colliding systems that we consider.

In the numerical setting, round-off somewhat ameliorates this issue and the iterative process
generally terminates \cite{Chatterjee:1998}. Relying on round-off is not acceptable, however. Moreover, we observe that convergence behavior consistently worsens in proportion to the decrease in $c_r$ (see Fig. 2.4 (d)). While iterating the generalized-reflection operator with a $c_r < 1$ would require fewer iterations than pairwise propagation, GR is nevertheless a propagation approach, and so it would not escape inelastic collapse.

**An energetic restitution model safe from inelastic collapse** Instead of iteratively re-solving impacts at $c_r < 1$ and suffering the consequences of inelastic collapse, we propose a simple and, to our knowledge, novel energetic restitution model that obtains consistent convergence and dissipation behaviors across all $c_r$ values. We first observe that purely inelastic multi-impact is well-posed and soluble using the standard LCP formulation \cite{Moreau:1983b,Anitescu:1997,Stewart:2000a}, i.e., (2.3) with $c_r = 0$, yielding $\dot{q}^+ = 0$. Similarly, elastic multi-impact is unaffected by collapse; applying the generalized reflection operator to the elastic case we obtain $\dot{q}^+_1$. We view $c_r$ as the interpolant between the two and thus obtain

$$\dot{q}^+ = (1 - c_r)\dot{q}^+_0 + c_r\dot{q}^+_1.$$

Notice that this definition of $c_r$ now gives the exact interpolation between the maximum (physically) allowable dissipation and total conservation of energy. In the special case of an isolated impact, we recover the classical isolated restitution model (2.2).

By construction this interpolation satisfies (KIN), (MOM), (ONE), (SYM), and (BRK). Feasibility follows from the linearity of the impact constraints.

### 2.6 Friction

As in contact, friction is a critical and often dominant aspect of transient collision behavior \cite{Brogliato:1999}. We incorporate impulsive friction (Alg. 3) while preserving the physical desiderata. The short time-scales of impact simplify the computation.

The impact solution, $\lambda$, gives the magnitude of the normal impulse and thus, in analogy to frictional contact, defines local, pointwise Coulomb inequalities, $\|f_k\| \leq \mu_k \lambda_k$, for friction impulse feasibility at each collision point $k$. In the instantaneous setting, as with normal directions, the directions of tangential dissipation are effectively fixed and thus the maximal dissipation of
friction \cite{Stewart2000} is reduced to impulses applied along a single \textit{fixed} tangential direction per collision point.

Instantaneously we have

\[
\dot{q}^+ = \dot{q}^- + M^{-1}G\lambda + M^{-1}D\beta,
\]  

(2.4)

where \( \lambda \) is the impact solution, \( D \) is the generalized basis of instantaneous friction directions that oppose velocity at each collision point, and \( \beta \) is the vector of corresponding frictional impulse magnitudes. Maximal dissipation then gives the frictional impulses as

\[
\beta = \arg\min_z z^T D^T M^{-1} (\dot{q}^- + M^{-1}G\lambda + M^{-1}Dz) \quad \text{s.t. } 0 \leq z \leq \text{diag}(\mu)\lambda.
\]  

(2.5)

To compute the friction basis \( D \) we let \( \Gamma_k \) denote the relative velocity Jacobian \cite{Kaufman2008} so that \( v_k = \Gamma_k \dot{q}^- \in \mathbb{R}^3 \) gives the relative velocity at collision point \( k \). Extracting the relative velocity local tangent component, \( v_{t,k} \), we then generate a \textit{single} friction basis contribution per collision point, \( d_k = \Gamma_k^T v_{t,k} / \|v_{t,k}\| \). The generalized friction basis follows as \( D = (d_1, ..., d_m) \).

Solving maximal dissipation globally guarantees monotone energy decay at all active contacts thus ensuring (KIN), our tangent basis choice preserves (MOM), while \textit{basis independence} continues to maintain (SYM). We then inherit (BRK) and (ONE) from GR.

\textbf{Algorithm 3} Friction\((q,p,A,\lambda)\)

\begin{enumerate}
\item \( G \leftarrow G_k(q) \)
\item \( D \leftarrow D_k(q,p) \)
\item \( \beta \leftarrow \arg\min_z (z^T D^T M^{-1}(p + G\lambda + Dz) : 0 \leq z \leq \text{diag}(\mu)\lambda) \)
\item \textbf{return} \( \beta \)
\end{enumerate}

\section{2.7 Numerical Implementation}

Our method requires the solution of numerous LCPs. Resolving even a single, large-scale LCP has long been considered computationally burdensome and has thus motivated many approximations and failsafes \cite{Erleben2007, Harmon2008}. We follow the observation that each
impact LCP problem corresponds to the optimality conditions of a convex Quadratic Program (QP) [Boyd and Vandenberghe, 2004]. In particular, we solve the large-scale, sparse impulse QP, dual to the LCP, employing an interior-point solver with a swappable linear backend. As we will see in §3.5 this leads to practical timings for exact LCP solves, even for very large (> 2M DoFs and > 2M constraints) impact systems.

**QP solution** The constraint gradients forming the linear constraints at each such solve are highly sparse\(^2\). To exploit the sparse problem-structure of these QPs we employ an interior-point method. Here the computational crux is to robustly and efficiently solve the many repeated inner-loop linear, KKT systems [Boyd and Vandenberghe, 2004]. We use the Ipopt solver [Wächter and Biegler, 2006] where the ability to customize our choice of an adaptive, swappable suite of linear-solver backends far outweighs the overhead of employing a general-purpose code. In particular, we employ MA27 [HSL, 2001] as our first-line, linear solver with MUMPS [Amestoy et al., 2001; Amestoy et al., 2006] invoked in the rare event that MA27 fails to solve a linear system.

For the impact QP, we observe that imposing non-negativity as a bound constraint yields a significant performance increase over imposing non-negativity as an affine constraint. In the case of a single friction disk sample, the Coulomb constraint similarly reduces to a bound constraint, which therefore admits the same optimization.

**QP solvers and scalability** While scaling our method to larger simulations, we tested a number of algorithms for solving the impact and friction QPs. For QPs with order 100 constraints, we find that the QL [Schittkowski, 2005] implementation of the Goldfarb and Idnani [Goldfarb and Idnani, 1983] dual active set method performs admirably. In fact this is the method we used for the small simulations in section 2.9. QL is a dense method, however, and as our simulations approach order 1,000 constraints, storing the normal equations in a dense manner grows computationally prohibitive.

For QPs of order 1,000 to 10,000 constraints, we tested an operator-based non-negative least-squares (NNLS) [Lawson and Hanson, 1974] approach that avoids the normal equations

\(^2\)Non-zeros in each column are generally restricted to the DoF stencil of any two in-contact surface patches.
altogether. We employ a two-metric, projected-descent solver [Friedlander, 2007] that enables warm starting and requires only the evaluation of operator-based callbacks for multiplications of impulse subvectors by submatrices of the constraint gradient. As we scale to larger contact problems, however, the NNLS approach requires increasingly aggressive applications of Tikhonov regularization, degrading overall performance compared to the interior point approach.

2.8 Finite Time Integration

While the instantaneous story reveals the inherent beauty of the underlying physics, any practical application must consider integration over finite amounts of time. The extension to handle this poses real challenges (§2.8.1). We address some of these, which leads us to propose a collision time-integration algorithm (§2.8.3) that is applicable to rigid body simulation and preserves all of the desiderata presented earlier. To validate the method we consider a wide range of benchmark examples (§9, §10, and §11). Finally, looking forward, we conclude with a discussion of limitations and open questions that this work raises (§12).

2.8.1 From One Instant to Finite Time

Prior to this point, our intentionally myopic discussion has focused exclusively on a single instant in time. We have developed an instantaneous multi-impact operator that, for the first time, is able to fulfill five important physical principles that apply to instantaneous impact. A natural question is what happens in the surrounding interval of time, and in particular whether an algorithm that satisfies the five principles of instantaneous impact fares better than other algorithms when employed as one component of a finite time integrator. In the remainder of this paper we explore this question.

As in the instantaneous picture, finite time behavior can be studied in terms of break-away, symmetry, momentum, the no-velcro condition, and energy conservation or boundedness. The last criterion however raises a subtle point.
2.8.2 Finite Time Energy Conservation

To simplify the discussion consider purely elastic restitution \( c_r = 1 \). In the instantaneous picture, positions were fixed, only momenta varied, and we sought exact conservation of kinetic energy. In the finite time picture, both configurations and momenta evolve, and we consider instead the requirement of exact conservation of the Hamiltonian, or total energy, over extended durations of simulation, which we denote (HAM). For inelastic impact \( c_r < 1 \) we relax (HAM) to allow for bounded rather than conserved total energy.

Do impact operators satisfying (KIN) lead to integration algorithms satisfying (HAM)? When we combine instantaneous impact response with finite time integration, we must distinguish between three related notions of “energy behavior:"

- where the long-term Hamiltonian conservation of the discrete time integrator alone (without impact response) obeys (HAM);
- whether the impact response instantaneously obeys (KIN);
- where the long-term Hamiltonian conservation of both pieces combined obeys (HAM).

If either the integrator has poor energy behavior, or the impact response violates (KIN), it is exceedingly unlikely that their aggregate will have good behavior. On the other hand, if both exactly conserve energy, their combination will conserve it as well. In the remainder of this paper, we explore the latter case, identifying two classes of physical systems where it is possible to combine GR with energy-momentum preserving integrators. We then discuss the inherent limitations and posit further avenues for exploration.

2.8.3 Synchronous Time Integration

In our finite time exploration we consider the typical case of synchronous or fixed-time-step collision integration. At the beginning of each time-step we gather all constraints expected to be violated along the course of a fixed-size time-step and then resolve all of them instantaneously and simultaneously. We do this by applying our chosen method (GR, Gauss-Seidel, Jacobi, or LCP) for Elastic Impact interpolated, by our restitution model, with Inelastic Impact. Friction is then applied, followed by a fixed time-step with an unconstrained, free-flight inte-
grator, denoted FF\_Integrator. The resulting collision-integration method is given below in Alg. 4.

While the solution of the maximal dissipation problem in impact remains \cite{Kaufman2005} as with contact coupled to constraint resolution forces \cite{Kaufman2008}, we will assume that most of the impacts are sufficiently high-speed that it is not critical to find the exact equilibrium between friction and contact forces. As such we discretize in time by applying a single pass sequence of GR followed by a maximal dissipation solve of the resulting QP in equation \eqref{eq:QP} above with $\lambda$ given. If a more accurate friction solution is needed then additional staggered iterations can be made \cite{Kaufman2008a}.

\begin{algorithm}
\caption{Collision\_Integrator($q^t,p^t,h,c_r$)}
\begin{algorithmic}[1]
\State $(\tilde{q},\tilde{p}) \leftarrow$ FF\_Integrator($q^t,p^t,h$)
\State $A \leftarrow \text{Get\_Active\_Set}(\tilde{q})$
\State $\lambda_1 \leftarrow \text{Elastic\_Impact}(q^t,p^t,A)$
\State $\lambda_0 \leftarrow \text{Inelastic\_Impact}(q^t,p^t,A)$
\State $\lambda \leftarrow c_r \lambda_1 + (1 - c_r) \lambda_0$
\State $\beta \leftarrow \text{Friction}(q^t,p^t,A,\lambda)$
\State $p^+ \leftarrow p^t + G_A(q^t)\lambda + D_A(q^t,p^t)\beta$
\State $(q^{t+1},p^{t+1}) \leftarrow$ FF\_Integrator($q^t,p^+,h$)
\State \textbf{return} $(q^{t+1},p^{t+1})$
\end{algorithmic}
\end{algorithm}

\section{Case Study: 2D Billiard Balls}

As a didactic starting point we consider frictionless billiard balls (two translational DoFs) in two dimensions under linear potentials. We employ Verlet \cite{Hairer2002} time integration, which for this physical system exactly conserves both momentum and energy. We consider friction in section \ref{sec:friction}. The ball-ball non-overlap constraint between 2D balls is easily formulated analytically. We compare Jacobi, Gauss-Seidel, Generalized Reflections, and LCP for elastic impact problems.


Figure 2.5: The “cleanest” pool break: A perfectly symmetric, elastic pool break is simulated for 56 balls, starting from symmetric initial conditions. After ten seconds of simulation time GR retains the initial symmetries while Gauss-Seidel orderings all generate unexpected, order-dependent results (the right-hand figure shows one representative Gauss-Seidel simulation).

**Symmetry in multi-impacts** During a clean pool break, racked balls disperse across the pool table in a nearly symmetric pattern. We consider the “cleanest” possible pool-break, i.e., the outcome of an impact subject to perfectly symmetric initial conditions. The initial setup for this simulation is an exactly aligned rack of 55 balls with an additional cue ball fired along the axis of symmetry. In Fig. 2.5 we compare the results after ten seconds of simulation time between GR and a Gauss-Seidel ordering.

**Breaking contact in multi-impacts** Consider a uniform grid of balls colliding with a circular boundary. Unless the impact is perfectly inelastic, the grid is expected to scatter with a fountain-like trajectory (see Fig. 2.6 top). Resolution of breaking contact is essential to produce this expected behavior. Indeed, simulating this system with LCP, and thus losing breaking contact, produces the unexpected behavior of the entire mass of balls bouncing back up, cohering together as a uniform, sticky material (see Fig. 2.6 bottom).

**Long-term kinetic energy conservation** In Fig. 2.8 we confirm the exact kinetic energy conservation of our collision integrator using LCP, Gauss-Seidel, and GR as impact operators and note the characteristic poor energy behavior of Jacobi. Here we plot the energy of a 9x9
Figure 2.6: Sticky impacts: As in the 1D case, LCP’s inability to resolve breaking contact introduces noticeable sticking artifacts in simulation (bottom). Compare to the breaking splash generated by GR (top).

grid of balls, constrained in a drum, given initial random velocities, and stepped at $h = 10^{-2}$ in zero gravity.

**Long-term symmetry conservation**  We place four balls in a circular enclosure with initial state symmetric about the vertical bisector, and plot trajectories generated with the GR, LCP, Jacobi, and Gauss-Seidel impact operators in the collision integrator (see Fig. 2.7). In this example LCP does not exhibit sticking, therefore the trajectories of GR and LCP coincide. Observe GR/LCP’s long-term symmetry preservation. The GS trajectory breaks symmetry upon the first impact. Jacobi’s trajectory preserves symmetry, but rapidly dissipates to a crawl, evidenced by the short trace length.

### 2.10 Case Study: 3D Rigid Bodies

We now shift our focus to the simulation of rigid bodies with friction in three dimensions. For free-flight integration in SE(3) with a linear gravitational potential we apply the Discrete Moser-
Veselov (DMV) integrator [Moser and Veselov, 1991], which is exactly energy, momentum, and angular-momentum conserving for this special case. In our implementation we employ a freely available implementation of DMV [Hairer and Vilmart, 2006]. Constraints are obtained for boxes using the standard box-box routine [Smith, 2006], while constraints for potentially non-convex mesh-mesh impacts are obtained with the signed distance field based implementation in the freely available OpenTissue package [Erleben and Dohlmann, 2007]. As an acceleration broad-phase culling is performed using a uniform spatial partition [Ericson, 2004].

2.10.1 Long-Term Energy Conservation

When married to a discrete time integration method, the choice of an instantaneous impact technique can lead to a harmonious partnership or a short-lived and explosive affair. In Fig. 2.9 we plot the long-term total energy of DMV when paired with Jacobi, Gauss-Seidel, and Generalized Reflections for three select simulations. Jacobi’s failure to respect (KIN) quickly destabilizes the entire simulation, leading to a catastrophic failure. Gauss-Seidel and Generalized Reflections, in contrast, act in concert with DMV to yield constant Hamiltonians and stable simulations.

2.10.2 Long-term Symmetry Preservation

Similarly, pairing DMV with the wrong instantaneous impact technique can send a simulation on an asymmetric and meandering trajectory. In Fig. 2.9 we plot traces of points on rigid bodies for the same three simulations. The trajectories produced by DMV with Jacobi and with Generalized Reflections mirror all spatial symmetries in the initial conditions, while the trajectories due to Gauss-Seidel quickly wander into disorder.

2.10.3 Long-term Angular Momentum Conservation

To verify conservation of (MOM), we place two bunnies in a simulated teapot with a coefficient of restitution of $c_r = 0.8$ and a coefficient of friction of $\mu = 0.5$ and initialize the system with zero net linear momentum and nonzero net angular momentum. As this system is closed, we expect both linear and angular momentum to remain constant. Indeed, evolving the system with GR and DMV using a time-step of $h = 0.01$ we observe a maximum drift of $4.37 \times 10^{-11}$
percent in each component of angular momentum and an absolute drift of $1.36 \times 10^{-11}$ in each component of the linear momentum for 1000 seconds of simulation.

### 2.10.4 Controlled Dissipation

Our generalized restitution model, when paired with a discrete time integration method, yields a predictable and controlled rate of dissipation. To demonstrate this, we simulate 1,000 bunnies in a fixed container with gravity using Generalized Restitution and DMV. As we decrease the coefficient of restitution, we observe an attendant controlled decrease in the long term energy dissipation (Fig. 2.10).

#### 2.10.5 Break-Away and Restitution

Neglecting (BRK) can produce unexpectedly uneventful simulations, even for systems subject to restitution effects. Dropping a set of matryoshka dolls with $c_r = 0.8$, the LCP model causes the dolls to rebound as a single, solid object (Fig. 2.2, left doll). With the Generalized Restitution model, in contrast, the dolls separate upon first impact, revealing the interior matryoshkas (Fig. 2.2, right doll).

### 2.11 Scaling and Experimental Benchmarks

We now evaluate the scaling properties of our method in 2D and 3D and benchmark against classical problems in granular media.

#### 2.11.1 Related Methods in Granular Simulation

While our proposed algorithms enable the simulation of arbitrary rigid body models, many of the following examples particularly focus on granular systems. Such examples enable us to consider correctness, efficiency, scaling, and emergent behavior in assemblies (treated here as collections of rigid bodies) while allowing us to validate large-scale simulations against experimental observations.
Table 2.2: Timings: Performance statistics for balls constrained to a drum. For each simulation we report the wall-clock time (seconds) spent in unconstrained integration, collision detection, and impact response per time-step. We report the min, max, and average time across all time-steps. We also report the number of iterations spent in each GR solve, as well as the overall active set size. The system was integrated with a fixed time-step of 0.01s and employed the nonlinear optimization package Ipopt to solve the impact QP. Timings were recorded with a single thread on a 3.33GHz Intel Core i7-975.

The simulation of granular materials has been broadly treated in engineering and mechanics. Pöschel and Schwager [Pöschel and Schwager, 2005a] provide a comprehensive survey of current methods in the literature.

Within the field of graphics research on granular simulation has largely focused upon continuum-based models [Zhu and Bridson, 2005a; Lenaerts and Dutre, 2009; Narain et al., 2010; Alduán and Otaduy, 2011a] that efficiently and convincingly capture the complex behaviors of granular flow. However, as noted by Narain et al. [Narain et al., 2010], these methods are unable to resolve the collisional behaviors of granular systems.

Alternately, a range of damped-spring-based interaction methods [Miller and Pearce, 1989; Luciani et al., 1995a; Bell et al., 2005a; Alduán et al., 2009a], originating in the discrete element and molecular dynamic models of mechanics, have also been considered. While effective for resolving slower contact modes, under impact these methods must often deal with many of the same stability issues commonly encountered in comparable, and more familiar, penalty-based methods Pöschel and Schwager, 2005a.
2.11.2 Scaling

Formulating our multi-impact problem as a QP and solving it in a sparsity preserving manner enables our method to scale to large problems composed of many DoFs in impact with one another. We instrumented two example configurations, consisting of increasingly larger numbers of frictionless spheres, all with unit radius, subject to $c_r = 1$, and constrained to stay inside a circular drum:

*Drop* examples are initialized to a uniform square grid configuration and dropped from rest, under gravity.

*Gas* examples are likewise initialized to a uniform square grid configuration. Each sphere is assigned an initial, random, unit length velocity. No external forces are exerted on the system.

In both examples, note the preservation of large scale symmetries. Timings and statistics for these examples are reported in Table 2.2. A video of these simulations can be found in the supplementary materials while configuration snapshots are shown in Fig. 2.11. Note that the dominant cost, the solution of the impact QP, scales with the number of constraints.

2.11.3 Granular Maxwell’s Demon

Dissipative impact plays an intriguing role in spontaneous symmetry breaking. An interesting example is the so-called Granular Maxwell’s Demon. In an analogy to Maxwell’s Demon, two identical grids of balls are placed in a box separated from one another by a partial-height partition. Driving the box’s floor at constant frequency and amplitude all balls eventually “choose” one of the two box partitions *without* the assistance of a demonic agent.

The standard explanation is that multiple dissipative collisions “trap” particles preferentially on the side with more particles [van der Weele *et al.*, 2001]. More particles colliding means more dissipation due to the coefficient of restitution, keeping the kinetic energy of the balls on the more populous side low. Similarly, balls on the sparser side experience a lower collision rate and thus their energy is not damped as quickly. Somewhat paradoxically, balls on the sparser side are thus more likely to cross over to the trapping side than vice-versa; as the trapping side becomes more loaded, this process accelerates via positive feedback.

Duplicating the experimental setup’s initial configuration we simulate the same effect using Generalized Reflections. Fig. 2.12 shows simulation snapshots of the granular Maxwell’s Demon.
phenomenon. With initially symmetric positions and random initial velocities, progressively more balls migrate to one side (the right in this example) as expected. This provides one validation of our proposed definition for the multi-impact coefficient of restitution.

2.11.4 Extended Patterns

Molecular and crystalline assemblies play a critical role in systems of interest in biology, chemistry and physics but are often difficult and costly to study. To understand these phenomena scientists have turned to simpler systems such as vibrated bins of granules [Umbanhowar et al., 1996]. In this setting spontaneous, stable patterns are observed [Melo et al., 1994]. The hope is that these systems can be studied in analogy to crystalline and molecular structures and thus elucidate otherwise complex behaviors.

The so-called extended patterns are emergent behaviors known to be driven solely by floor vibrations, high-speed inelastic impacts between granules, and corresponding frictional forces [Melo et al., 1994]. As such they are ideal computational benchmarks to examine both validation and efficiency. The challenges to simulate them are directly in line with our goals: the generation of extended patterns requires the accurate resolution of very-large collections of rigid-bodies under high speed impact (and are thus not amenable to continuum models), they require the accurate, scalable modeling of multi-impact (simultaneous collisions are generic), restitution (inelasticity is required), and friction (without friction the patterns do not emerge [Moon et al., 2004]) at large scales.

We first confirm that our algorithm captures the stable stripe, square and hexagonal patterns experimentally observed in stability regions of a phase diagram plotting driving amplitude vs. frequency. Duplicating the reported [Bizon et al., 1998] material restitution and friction, packing ratios, amplitudes, and frequencies we obtain a one-to-one correspondence with experimental results [Bizon et al., 1998] for all three patterns: hexagons at $f^* = 0.38$ and $\Gamma = 4.00$, squares at $f^* = 0.27$ and $\Gamma = 3.00$, and stripes at $f^* = 0.44$ and $\Gamma = 3.00$ (see Fig. 2.13). These parameters correspond to the non-dimensionalized frequency and amplitude of the driving plate, which are employed to reduce the dimensionality of the problem search space. Note, as well, as seen in our supplemental video, that these extended patterns emerge, as expected, dynamically over time and remain stable throughout the remainder of all simulation runs in correspondence with
CHAPTER 2. STRUCTURE PRESERVING IMPACT SIMULATION

Table 2.3: Hexagonal extended pattern statistics: Performance for decreasing time-step sizes. Timings (reported above in seconds) were recorded with a single thread on a 2.67GHz Intel Xeon 5650 for a system of 360,000 DoFs.

the experimentally observed stability.

Next we note that these patterns are simulated “out of the box” in a parameter-free manner with no tuning. However, since we employ a synchronous framework for simulation (see [2.8.3]) in these examples, there is a clear trade-off between the accuracy of our results, e.g., how well we capture the extended patterns, and the size of the time-step employed, i.e., the efficiency of the method. In Table 2.3 we consider these trade-offs in the simulation of the hexagonal pattern holding all experimental constants fixed and varying only time-step size in increments down from $h = 10^{-3}$ (above which only a flat “noisy” pattern is obtained) decrementing down to $h = 10^{-4}$. At $h = 10^{-3}$ the expected hexagonal pattern emerges; however, the wavelength is smaller than reported. As we decrease time-step the wavelength of the generated hexagonal pattern correspondingly increases until, starting at step sizes of $h = 3 \times 10^{-4}$ and below, simulations converge to both the correct pattern and wavelength.
Figure 2.7: Long-term symmetry: Long-term trace of particle trajectories for GR/LCP, Gauss-Seidel and Jacobi. GR, LCP, and Jacobi remain symmetric, while Gauss-Seidel breaks symmetry. Note that LCP does not suffer from ‘sticking’ for this example, and so its solution does not differ from GR’s. Jacobi’s trajectory quickly damps out due to parasitic dissipation.

Figure 2.8: Long-term kinetic energy: Over a finite time interval, impact response using Jacobi (solid) yields a random walk type energy behavior, which eventually destabilizes the simulation. GS, GR, and LCP (dashed) exactly conserve kinetic energy, for this system.
Figure 2.9: (HAM) and (SYM) with DMV: Here we show the results of three simulations. Top: Eight cubes, initially touching, with symmetric outward velocities constrained to lie in a large cube. There is no gravity in this simulation. Middle: A trefoil knot with threefold rotational symmetry is dropped on a plane under gravity. Bottom: Two bunnies with 180 degree rotational symmetry are released under gravity in a box. Left: For each simulation, we plot traces of points fixed on each body. Right: For each simulation, we plot potential (blue), kinetic (green), and total energy (red). Long term desiderata: Coupled with a time integrator, Jacobi, to the left, respects (SYM) but drops (HAM) while Gauss-Seidel, in the middle, conserves (HAM) but destroys (SYM). Only GR, on the right, satisfies both (HAM) and (SYM) when paired with DMV. To avoid ambiguity in selecting a vertical scale to display the Hamiltonian with, we employ the natural scale defined by the (always positive) kinetic energy.
Figure 2.10: Controlled dissipation: As we decrease $c_r$, the overall dissipation decreases in a controlled manner.

Figure 2.11: Scaling of our system: Here we show two sequences of configurations from large-scale, 1M-ball simulations in 2D. Impacts are frictionless and elastic (i.e., $c_r = 1$). Top (Drop): Spheres are initialized to a uniform square grid configuration, dropped from rest, under a gravitational potential of magnitude 10. Bottom (Gas): Spheres are initialized to a uniform square grid configuration. Each sphere is assigned an initial, random, unit length velocity. No external forces are exerted on the system.
Figure 2.12: Maxwell’s demon demonstrates a dramatic consequence of restitution behavior (left to right, top to bottom). Balls placed evenly in two sides of a vibrating box, split by a solid fence, aggregate on one side due to dissipative collisions. The setup here simulates 288 unit mass steel balls with $c_r = 0.6$. 
Figure 2.13: Extended patterns experiment and GR simulation: We simulated the experiments of Melo et al. [Melo et al., 1994] across a range of parameters. Top: In the left and middle columns we show side and top view simulation snapshots and note a match with the experimentally obtained square, stripe, and hexagonal patterns in the right column. Bottom: We also reproduce the phase diagram from Bizon et al. [Bizon et al., 1998] showing the points that generated both the experimental results and the matching simulations.
Chapter 3

Hybrid Simulation of Granular Materials

Formulating a general and scalable model for granular materials presents many challenges unique to the medium. Granular materials often behave as a continuum, with recent advances expanding the scope of phenomena that can be safely treated as a continuum. Despite these advances, individual grain-scale interactions are known to influence the evolution of bulk portions of the domain, leading to phenomena like frictional jamming and shear banding that are difficult to capture with a continuum model. Beyond the scientific importance of understanding these phenomena, in computer animations we desire finely resolved motions of individual grains at free surfaces, but interior regions of the material are less important to visualize, and can feasibly be treated with a continuum.

We propose to bridge the gap between continuum and discrete treatments of granular materials with a hybrid simulation method that inherits benefits from both worlds. We first derive an Arlequin-like approach to constrain the dynamics of a discrete model to a continuum model. We then discretize these constrained systems, revealing a family of possible discrete integrators for hybrid granular simulations. By choosing an explicit, penalty-based discrete element treatment of individual grains, and an explicit material point treatment of the continuum, we arrive at a predictor-corrector integration technique that is able to simulate hybrid systems. We then demonstrate that by treating discrete grains as material points in the constraint solve, that
we can completely eliminate all linear solves from the integrator. Finally, to support systems undergoing massive plastic deformations, as is typical in many flowing granular systems, we propose a technique to identify and update which regions of the material are treated with a discrete model, and which regions are treated with a continuum model.

3.1 Related Work

We draw on a wide body of literature on the modeling and simulation of granular materials with discrete and continuum treatments. We also draw inspiration from work on hybridizing other physical systems, particularly crystalline solids.

**Discrete particle simulations.** *Discrete particles*, originating in the molecular dynamics models of Alder and Wainwright [Alder and Wainwright, 1957, Alder and Wainwright, 1959, Alder and Wainwright, 1960], have a long and successful history across scientific and engineering disciplines [Hoover, 1986, Rapaport, 2004, Frenkel and Smit, 2001]. These methods explicitly model the behavior of individual particles and their subsequent interactions, and have been successfully extended as discrete element methods to simulate granular materials [Cundall and Strack, 1979, Haff and Werner, 1986, Walton and Braun, 1986, Gallas *et al.*, 1992, Pöschel and Schwager, 2005b]. These methods can be augmented to permit inelastic grain-level mechanics such as grain breakage/fragmentation, as arises in many complex real-world situations [Nguyen *et al.*, 2015, Åström and Herrmann, 1998, Tsoungui *et al.*, 1999, Ben-Nun *et al.*, 2010]. Discrete particle approaches employ either a “soft” or “hard” contact and friction formulation.

**Penalty methods** define a “soft” contact level constitutive relation and integrate the resulting forces. They are popular owing both to their simplicity and flexibility in defining interaction properties [Shäfer *et al.*, 1996, Kruggel-Emden *et al.*, 2007]. Penalty’s disadvantages lie in setting penalty parameters that often require multiple tuning and calibration passes, stiff, parasitic forces that are difficult to integrate, and challenges in modeling static piles and incompressibility. The graphics community has explored penalty-based simulations of granular materials, with works focused on the formulation of the constitutive model [Miller and Pearce, 1989, Luciani *et al.*, 1995b], approximations of static friction through irregularly shaped grains [Bell *et al.*, 2005b], resolution up-sampling [Alduán *et al.*, 2009b], extensions to position-based sim-
ulations [Macklin et al., 2014], and interactions with fluids [Rungjiratananon et al., 2008].

Contact dynamics (CD) methods treat interactions with “hard” constraints [Baraff, 1989b; Jean, 1999a; Brogliato, 2012; Acary and Brogliato, 2008; Stewart, 2011]. Originating in the work of Moreau [Moreau, 1983a; Moreau, 1988b], CD treats contact and friction through constrained formulations, typically equivalent to mixed linear complementarity formulations (MLCP) [Stewart and Trinkle, 1996; Stewart, 2000b]. While CD avoids many drawbacks of penalty methods, finding optimal solutions to their MLCP formulation is computationally difficult [Kaufman et al., 2008b]. Significant attention has been directed to the accurate numerical solution of these models [Alart and Curnier, 1991; Stewart, 2001; Duriez et al., 2006; Bonnefon and Daviet, 2011; Daviet et al., 2011; Jourdan et al., 1998; Jean and Moreau, 1992], to convex relaxations of CD [Anitescu and Hart, 2004; Mazhar et al., 2015; Preclik, 2014], to numerical methods suited for interactive simulations [Erleben, 2007b; Tonge et al., 2012], and to non-zero restitution [Smith et al., 2012; Uchida et al., 2015].

Continuum Granular Models. Our work builds on a rich history in the study of granular materials, the foundations of which were laid by Coulomb who first posited a relationship between the imposed pressure and the resistance to shear motion through a coefficient of friction. Modern variants include critical-state [Schofield and Wroth, 1968] and anisotropic models [Rothenburg and Bathurst, 1989; Dafalias et al., 2004]. The rate-sensitive inertial rheology [G.D.R. MiDi, 2004; Jop et al., 2006] has proven effective at modeling fast flows, while non-local models [Mohan et al., 2002; Aranson and Tsimring, 2002; Kamrin and Koval, 2012] account for finite grain sizes by introducing non-local terms to the continuum description.

Continuum Simulation Methods Within the scientific and engineering communities, a variety of simulation techniques for granular materials have been proposed, typically targeted towards specific flow regimes. For quasi-static and small-strain simulations, standard finite element analysis techniques are applicable [Zienkiewicz and Taylor, 2000]. Finite volume discretizations have been successfully applied to faster flow regimes [Lagrée et al., 2011]. Recently, Dunatunga and Kamrin [Dunatunga and Kamrin, 2015] presented simulations of granular materials using the Material Point Method (MPM) [Sulsky et al., 1994].

Within the graphics community the simulation of granular materials with hybrid Eulerian-Lagrangian discretizations has received significant attention. Zhu et al. augmented an incom-
pressible fluid-implicit particle (FLIP) solver with a Mohr-Coulomb yield criterion to identify rigid and flowing regions [Zhu and Bridson, 2005b]. Narain et al. draw an analogy to a discrete contact dynamics solver, and by mirroring the discrete solver’s structure obtain unilaterally incompressible granular simulations with a faceted frictional yield criterion [Narain et al., 2010]. Daviet et al. remedy artifacts in Narain’s approach by employing a smooth yield criterion and support more general models like the $\mu(I)$ rheology [Daviet and Bertails-Descoubes, 2016], while Klár et al. augment an elastoplastic MPM simulation with a Drucker-Prager yield criterion [Klár et al., 2016]. More recently, mixtures of sand and water have been simulated with the MPM framework [Tampubolon et al., 2017].

In concert with the developments in hybrid Eulerian-Lagrangian discretizations, the graphics community has pushed work on fully Lagrangian discretizations forward, beginning with incompressible smooth-particle hydrodynamics (SPH) techniques [Lenaerts and Dutr, 2009]. Subsequent work has explored unilateral incompressibility in the context of SPH [Alduán and Otaduy, 2011b]. Recent works have continued to incorporate general advancements in SPH, with impressive visual results obtained through force-aware upsampling [Himsen et al., 2013].

The graphics community has also explored the specialized simulation of granular surfaces [Li and Moshell, 1993; Chanclou et al., 1996; Sumner et al., 1999; Onoue and Nishita, 2003; Pla-Castells et al., 2008], achieving fast simulations for restricted modes of deformation.

**Hybrid Granular Simulations** The notion of hybridizing small- and large-scale simulation tools to capture the best aspects of both is a relatively new one. “Quasi-Continuum” and “Arlequin-type” methods have been explored primarily for crystalline solids, to expedite otherwise lengthy atomistic simulations by hybridizing with a crystal plasticity continuum model in zones where atomistic refinement is not needed [Tadmor et al., 1996; Smith et al., 2001; Shimokawa et al., 2007; Zhang and Ge, 2005; Dhia, 1998]. The idea of hybridizing discrete-particle and continuum approaches to simulate granular media is in its infancy, with only initial work done to show the validity of communicating mechanics between discrete grains and finite-element facets [Yan et al., 2010]. Recent work has explored when continuum and discrete treatments are simultaneously accurate [Rycroft et al., 2009; Kamrin, 2010a; Kamrin and Koval, 2014], including an Arlequin-type method that couples statically-defined regions of a DEM simulation to the interior of a continuum FEM-based simulation to enrich stress fields.
around drill tips, for instance [Wellmann and Wriggers, 2012]. In contrast, we target regimes in which enriched degrees of freedom are required at surfaces, and where the boundary between continuum and discrete regimes evolves dynamically.

In the granular physics and graphics literature, lower-level ideas have been tried where instead of implementing a general continuum model, the user imposes kinematic constraints to the particle motion in certain regions, often chosen based on experience with the problem at hand. The graphics literature has explored freezing rigid bodies that are sufficiently stationary [Smith, 2005]. Similar techniques have been proposed to accelerate the generation of granular packings for industrial applications [Mio et al., 2009]. In common granular setups such as rotating tumblers and growing sand piles, semi-empirical models can be used to guess zones of rigid material, and grains in these zones can be removed from the discrete update [McCarthy and Ottino, 1998; Hsu and Keyser, 2010; Zhu and Yang, 2010; Bouchaud et al., 1994]. Holladay et al. [Holladay and Egbert, 2012] carve out interior regions of granular materials moving at constant velocities and replace these groups of grains with meshes, but this method does not homogenize over rigidly rotating regions or over shear flows, and as the paper notes, can lead to volume loss. These ideas have been developed further in follow-up work [Holladay, 2013; Munns, 2015]. These methods make no claims as to the accuracy of the techniques for science and engineering applications, and have not yet demonstrated stable granular flows.

### 3.2 Hybrid Kinematics and Dynamics

Our ultimate goal is to couple two consistent discretizations of a granular medium that are tailored to different scales. To that end, we begin by (variationally) deriving a general method to spatially partition the equations of motion of a mechanical system into coupled, overlapping domains. With a simple constraint, the combined evolution of these systems exactly reproduces the behavior of the original, unpartitioned system. With this formalism in place, we will apply separate discretizations to the two partitioned domains, corresponding to a discrete and a continuum treatment. This development can be viewed as a variant of an Arlequin-type approach. Finally, we propose a method to identify regions that require a discrete treatment, and a method to dynamically transition between discrete and continuum treatments as a simulation.
3.2.1 Hybridization Through Constraints

Consider a system with density $\rho(x)$ defined over the reference coordinates $x \in \Omega$ of a body (Fig. 3.1). We partition the density of the system with a time dependent weight function $w(x,t) \in [0,1]$, ensuring that we recover the original density:

$$\rho(x) = w(x,t) \rho(x) + (1 - w(x,t)) \rho(x)$$

If we consider the new partitions to be separate systems with generalized coordinates $(q_0, v_0)$ and $(q_1, v_1)$, we can recover the kinematic description of the original system by demanding that $q_0 = q_1$. Note that while we could concatenate these coordinates to give a single generalized set of coordinates, we will keep them separate for pedagogical clarity. Also note that we omit explicit parameters when the dependence is clear.

Given that these two systems have the same initial configuration $q_0(x,t_0) = q_1(x,t_0)$, we can equivalently enforce equal velocities, giving the constraint:

$$c(x,t) = v_0(x,t) - v_1(x,t) = 0$$

For the remainder of this discussion we employ the velocity level constraint formulation, as this treatment will prove beneficial when we later discretize the system. Note, however, that
equivalent results follow with the position level constraint formulation. Consistent with figure 3.3, our ultimate goal is to treat the discrete particles and the continuum as the two subsystems with “reconciliation” zone wherever $0 < w < 1$.

Finally, observe that while our density weight function $w$ is defined over the reference space, we can treat spatial weight dependencies in the deformed configuration as time dependent terms via the reference map.

### 3.3 Dynamics

Before deriving the equations of motion for the coupled system, we will briefly review the Lagrangian reformulation of classical mechanics. The fundamental axiom of the Lagrangian reformulation, Hamilton’s principle, derives the equations of motion of a system from a quantity
$L = T - U + C$ known as the Lagrangian [Landau and Lifshitz, 1976; Bedford, 1985]. Here $T$ describes the system’s kinetic energy, $U$ describes the system’s potential energy, and $C$ describes any constraints acting on the system. A valid physical trajectory is a stationary path in the action integral $S = \int_{t_0}^{t_1} L dt$. That is, infinitesimal variations in the physical trajectory do not change the value of the action (Fig. 3.2). Starting from this single observation, one can derive the equations of motion of a constrained physical system using the calculus of variations. For dissipative systems, the analogous derivation follows from the Lagrange d’Alembert Principle, but the end result for our purposes is the same.

Before applying the calculus of variations, we need to define a suitable kinetic energy, potential energy, and constraint function for the Lagrangian. The kinetic energy of the coupled system is given by the sum of the kinetic energies from each isolated system:

$$T = \frac{1}{2} \int_\Omega \rho w \mathbf{v}_0^T \mathbf{v}_0 dV + \frac{1}{2} \int_\Omega \rho (1 - w) \mathbf{v}_1^T \mathbf{v}_1 dV$$

Similarly, the potential energy is given by the sum of the energy from each isolated system. We consider a specific energy $e(q)$ that yields the stored energy per unit mass. Gravity, for example, has a specific energy of $e(q) = -g^T q$, where $g$ is the acceleration due to gravity lifted to generalized coordinates. The total potential energy is thus:

$$U = \int_\Omega \rho w e(q_0) dV + \int_\Omega \rho (1 - w) e(q_1) dV$$

Finally, we introduce the coupling constraint to the system with Lagrange multipliers $\lambda$:

$$C = \int_\Omega \lambda^T (\mathbf{v}_0 - \mathbf{v}_1) dV$$

With these values in hand we can assemble the Lagrangian $L = T - U + C$ and, for any given trajectory, we can compute the action $S = \int_{t_0}^{t_1} T - U + C dt$. We seek a trajectory for which the action is stationary, so we will explore perturbations of the system of the form $q_0^* = q_0 + \epsilon \eta_0$, $q_1^* = q_1 + \epsilon \eta_1$, $\mathbf{v}_0^* = \mathbf{v}_0 + \epsilon \dot{\eta}_0$, and $\mathbf{v}_1^* = \mathbf{v}_1 + \epsilon \dot{\eta}_1$. Note that the perturbations must be 0 at the endpoints of the trajectory $\eta_0(t_0) = \eta_0(t_1) = \eta_1(t_0) = \eta_1(t_1) = 0$. Let $S^*(\epsilon)$ denote the action evaluated at a perturbed trajectory. We extremize the action by varying with respect to $\epsilon$ and equating to zero as $\delta S^*(\epsilon)|_{\epsilon=0} = 0$.

Starting with the kinetic energy, varying with respect to epsilon via the chain rule and
rearranging gives:

\[
\frac{\partial T^*}{\partial \epsilon} = \frac{\partial T^*}{\partial v_0^*} \frac{\partial v_0^*}{\partial \epsilon} + \frac{\partial T^*}{\partial v_1^*} \frac{\partial v_1^*}{\partial \epsilon}
\]

\[
= \dot{\eta}_0^T \int_\Omega w \rho v_0^* dV + \dot{\eta}_1^T \int_\Omega (1 - w) \rho v_1^* dV
\]

Plugging this result into the action integral, we next integrate by parts to shift the time derivative off of the perturbation terms:

\[
\int_{t_0}^{t_1} \frac{\partial T^*}{\partial \epsilon} dt = \int_{t_0}^{t_1} \left( \dot{\eta}_0^T \int_\Omega w \rho v_0^* dV + \dot{\eta}_1^T \int_\Omega (1 - w) \rho v_1^* dV \right) dt
\]

\[
= \left[ \dot{\eta}_0^T \int_\Omega w \rho v_0^* dV \right]_{t_0}^{t_1} - \int_{t_0}^{t_1} \left( \eta_0^T \int_\Omega w \rho a_0^* + \dot{w} \rho v_0^* dV \right) dt
\]

\[
+ \left[ \dot{\eta}_1^T \int_\Omega (1 - w) \rho v_1^* dV \right]_{t_0}^{t_1} - \int_{t_0}^{t_1} \left( \eta_1^T \int_\Omega (1 - w) \rho a_1^* - \dot{w} \rho v_1^* dV \right) dt
\]

\[
= \int_{t_0}^{t_1} \left( -\eta_0^T \int_\Omega w \rho a_0^* + \dot{w} \rho v_0^* dV - \eta_1^T \int_\Omega (1 - w) \rho a_1^* - \dot{w} \rho v_1^* dV \right) dt
\]

where \(a_0^*\) and \(a_1^*\) denote each system’s acceleration. The terms evaluated at \(t_0\) and \(t_1\) are 0 as \(\eta_0(t = t_0) = \eta_1(t = t_1) = 0\).

Varying the potential energy with respect to \(\epsilon\) gives:

\[
\frac{\partial U^*}{\partial \epsilon} = \frac{\partial U^*}{\partial q_0} \frac{\partial q_0}{\partial \epsilon} + \frac{\partial U^*}{\partial q_1} \frac{\partial q_1}{\partial \epsilon}
\]

\[
= \dot{\eta}_0^T \int_\Omega w \rho \frac{\partial e(q_0^*)}{\partial q_0^*} dV + \dot{\eta}_1^T \int_\Omega (1 - w) \rho \frac{\partial e(q_1^*)}{\partial q_1^*} dV
\]

Finally, varying the constraint formulation with respect to \(\epsilon\) gives:

\[
\frac{\partial C^*}{\partial \epsilon} = \frac{\partial C^*}{\partial v_0^*} \frac{\partial v_0^*}{\partial \epsilon} + \frac{\partial C^*}{\partial v_1^*} \frac{\partial v_1^*}{\partial \epsilon}
\]

\[
= \dot{\eta}_0^T \int_\Omega \lambda dV - \dot{\eta}_1^T \int_\Omega \lambda dV
\]

We next plug this result into the action integral and shift the time derivative off of the pertur-
bation terms through integration by parts:

\[
\int_{t_0}^{t_1} \frac{\partial C^*}{\partial \epsilon} dt = \int_{t_0}^{t_1} \frac{\partial C^*}{\partial \epsilon} \int_\Omega \lambda dV dt
\]

\[
= \left[ \eta_0^T \int_\Omega \lambda dV \right]_{t_0}^{t_1} - \int_{t_0}^{t_1} \left( \eta_0^T \int_\Omega \lambda dV \right) dt
\]

\[
- \left[ \eta_1^T \int_\Omega \lambda dV \right]_{t_0}^{t_1} + \int_{t_0}^{t_1} \left( \eta_1^T \int_\Omega \lambda dV \right) dt
\]

\[
= \int_{t_0}^{t_1} \left( -\eta_0^T \int_\Omega \lambda dV + \eta_1^T \int_\Omega \lambda dV \right) dt
\]

Putting the components together, we find:

\[
\delta S^*(\epsilon) \bigg|_{\epsilon=0} = \eta_0^T \left[ \int_\Omega -w \rho \mathbf{a}_0 - \dot{w} \rho \mathbf{v}_0 - w \rho \frac{\partial e(q_0^*)}{\partial q_0} - \dot{\lambda} dV \right]
\]

\[
+ \eta_1^T \left[ \int_\Omega - (1-w) \rho \mathbf{a}_1 + \dot{w} \rho \mathbf{v}_1 + (1-w) \rho \frac{\partial e(q_1^*)}{\partial q_1} + \dot{\lambda} dV \right]
\]

\[
= 0
\]

As the perturbation can be arbitrary, the terms in brackets must be 0 everywhere to guarantee that the expression is 0. We thus obtain the equations of motion:

\[
\int_\Omega w \rho \mathbf{a}_0 dV = -\int_\Omega w \rho \frac{\partial e}{\partial q_0} dV - \int_\Omega \dot{w} \rho \mathbf{v}_0 + \dot{\lambda} dV
\]

\[
\int_\Omega (1-w) \rho \mathbf{a}_1 dV = -\int_\Omega (1-w) \rho \frac{\partial e}{\partial q_1} dV + \int_\Omega \dot{w} \rho \mathbf{v}_1 + \dot{\lambda} dV
\]

Notice that the rightmost terms under the integral, given the constraint \( \mathbf{v}_0 = \mathbf{v}_1 \), are equal. If we relabel this total term as a new Lagrange multiplier \( \bar{\lambda} = \dot{w} \rho \mathbf{v}_0 + \dot{\lambda} = \dot{w} \rho \mathbf{v}_1 + \dot{\lambda} \), we can view the new constraint forces as absorbing the force due to the density gradient. Alternatively, as these systems are constrained to overlap and the constraint forces must be equal and opposite, the forces naturally cancel when the system is viewed as a whole. Our final, simplified equations of motion are thus:

\[
\int_\Omega w \rho \mathbf{a}_1 dV = -\int_\Omega w \rho \frac{\delta e}{\delta q_1} dV - \int_\Omega \bar{\lambda} dV ,
\]

\[
\text{Volume}_1 \text{ force}
\]

\[
\int_\Omega (1-w) \rho \mathbf{a}_2 dV = -\int_\Omega (1-w) \rho \frac{\delta e}{\delta q_2} dV + \int_\Omega \bar{\lambda} dV ,
\]

\[
\text{Volume}_2 \text{ force}
\]

(3.1)
which are subject to the coupling constraint $v_1 = v_2$. The coupling force, which acts equally and oppositely on the two systems to enforce the constraint, arises naturally from the calculus of variations, averting the formulation of ad-hoc communication models between the two systems.

If we sum the two equations and substitute in the coupling constraint, we recover the original equations of motion for the entire simulation domain. The weight function naturally defines a partition of unity for the masses and the energies, with smaller weight values corresponding to a system having less influence in a given region. Outside the reconciliation zone, (3.1) is simply the (usual) equations of motion for two independent systems.

We now replace these two abstract systems with a discrete particle system and a continuum system (Fig. 3.3). By ansatz, we require the divergence of the stress in the continuum domain to be compatible with the frictional forces in the discrete domain. This is realizable through to the so-called Christoffersen formula [Christoffersen et al., 1981], which relates the continuum stress to the discrete frictional contact forces (Figure 3.4). We employ a penalty formulation for the discrete frictional contact forces, while we model the continuum medium as a plastic material with a Drucker-Prager yield criterion. We discretize and simulate the continuum medium with the Material Point Method (MPM).

### 3.3.1 Discrete Particle Simulation
Consider a granular medium in which each grain is modeled as a rigid body. In 2D, each grain’s configuration is parameterized by three degrees of freedom: two for the center of mass, and one for the orientation. In 3D, each grain’s configuration is completely parameterized by six degrees of freedom, including three degrees of freedom for the body’s center of mass $x_k$ and three degrees of freedom for the body’s orientation $R_k$. Given $K$ discrete grains, we concatenate all degree of freedom into a single generalized configuration vector $q_d \in \mathbb{R}^{2K}$ in 2D, and $q_d \in \mathbb{R}^{6K}$ 3D. We can similarly build a generalized velocity vector $v_d \in \mathbb{R}^{3K}$ and a generalized mass matrix $M_d \in \mathbb{R}^{3K \times 3K}$ in 2D, and $v_d \in \mathbb{R}^{6K}$ and $M_d \in \mathbb{R}^{6K \times 6K}$ in 3D. The evolution of the system is now compactly described by

$$M_d a_d = f_d(q, v_d, t),$$

$$\dot{q}_d = \omega(q) v_d$$

where $a_d$ is the generalized acceleration, $f_d$ contains all forces acting on the system. In 3D the force includes the so-called ‘quadratic velocity’ term, and $\omega(q)$ is the linear map from infinitesimal velocity changes to infinitesimal configuration changes. Note that the later two quantities depend on the choice of coordinates used to encode each body’s orientation, with standard formulas available in the literature [Shabana, 2013].

To model contacts between bodies, we employ a discrete element force-based penalty model [Cundall and Strack, 1979]. This model is well studied, is able to model stable piles, frictional jamming, and stick-slip behaviors, and has extensive validation against experimental results. Of importance to our needs, the Cundall and Strack penalty model has been shown to agree with continuum predictions in the regimes where we seek to homogenize the discrete dynamics [Rycroft et al., 2009]. The penalty force computation is divided into two components, including a normal force to resolve inter-grain penetration, and a tangential force to model friction. The normal and tangential forces are in turn related by a Coulomb friction constraint.

Concretely, the normal force is given by $f_n = k_n d n - \gamma_n v_n$, where $d$ is the penetration depth.
at the contact, \( \mathbf{n} \) is the contact normal, \( k_n \) is the normal contact stiffness constant, \( \mathbf{v}_n \) is the relative velocity projected into the normal direction, and \( \gamma_n \) is the normal damping coefficient. The tangential frictional force is given by \( f_t = k_t \Delta s - \gamma_t \mathbf{v}_t \), where \( k_t \) is the tangential contact stiffness, \( \Delta s \) is the tangential anchor spring (described below), \( \gamma_t \) is the tangential damping coefficient, and \( \mathbf{v}_t \) is the relative velocity projected into the tangent plane. At the end of each time-step, we update \( \Delta s \) by integrating the tangential relative velocity at the contact. We then project any normal component out of \( \Delta s \), and rescale \( \Delta s \) so the Coulomb constraint \( f_t \leq \mu f_n \) is satisfied, where \( \mu \) is the coefficient of Coulomb friction.

### 3.3.2 Continuum MPM

We model continuum granular regions as an elastoplastic material with a Drucker-Prager plastic yield criterion. The evolution of the system is governed by the conservation of momentum

\[
\rho \frac{D\mathbf{v}}{Dt} = \nabla \cdot \mathbf{\sigma} + \rho \mathbf{f}_{ext}
\]

and the conservation of mass

\[
\frac{D\rho}{Dt} + \rho \nabla \cdot \mathbf{v} = 0,
\]

where \( \mathbf{\sigma} \) denotes the Cauchy stress tensor, \( \frac{D}{Dt} \) denotes the material derivative, and \( \mathbf{f}_{ext} \) denotes any external body forces (e.g. gravity). We use a multiplicative decomposition of the deformation gradient \( \mathbf{F} = \mathbf{F}^e \mathbf{F}^p \) where \( \mathbf{F}^e \) and \( \mathbf{F}^p \) denote the elastic and the plastic component of the deformation gradient, respectively. We denote the left Cauchy-Green strain by \( \bar{\mathbf{b}}_e = \det(\mathbf{F}^e) - \frac{1}{2} \mathbf{b}_e \). In 3D, the strain energy density becomes

\[
W = \frac{\kappa}{2} \left[ \frac{1}{2} (J^2 - 1) - \ln J \right] + \frac{1}{2} \mu (Tr[\bar{\mathbf{b}}^e] - 2)
\]

\[
\tau = \frac{\kappa}{2} (J^2 - 1) \mathbf{I} + \mu \text{dev}[\bar{\mathbf{b}}^e]
\]

where \( \bar{\mathbf{b}}^e = \det(\mathbf{b}^e)^{-1/2} \mathbf{b}^e \) is the volume preserving elastic left Cauchy-Green strain and \( J = \det(\mathbf{F}) \). In 3D, the strain energy density becomes \( W = \frac{1}{2} \kappa \left[ \frac{1}{2} (J^2 - 1) - \ln J \right] + \frac{1}{2} \mu (Tr[\bar{\mathbf{b}}^e] - 3) \) while \( \bar{\mathbf{b}}^e = \det(\mathbf{b}^e)^{-1/3} \mathbf{b}^e \).
To allow the granular medium to separate, we consider the medium to be one-sided in the following sense: the granular medium can resist compression, but not extension. We model this effect with a free-flow mode, similar to Dunatunga et al. [2015]. When we detect that the material is in extension, indicated by $\det(b^e) > 1$, we project the strain to $\det(b^e) = 1$ by assigning $b^e \leftarrow \det(b^e)^{-1/2} b^e$. The net result is zero pressure when the material is under extension.

Figure 3.5: Drucker-Prager parameter sweep: We explore the effect of the $\alpha$ parameter in the Drucker-Prager model by running a series of column collapse simulations with varying values of $\alpha$. Increasing $\alpha$ leads to steeper angles of repose in the resulting pile of grains. The ability to tune the repose angle allows us to match the gross behavior of a continuum simulation to a discrete simulation, which enables physically consistent hybrid simulations.

To model plastic flow, we begin with the Drucker-Prager yield condition

$$\Phi = s - \alpha p \leq 0,$$

where $s = \|\text{dev}[\tau]\|_F$ is the magnitude of the shear stress (recall that $\text{dev}[X] = X - \frac{1}{2} \text{Tr}[X]I$), $p = -\frac{1}{2} \text{Tr}[\tau]$ is the pressure ($p = -\frac{1}{3} \text{Tr}[\tau]$ in 3D), and $\alpha$ controls the angle of repose. It often proves useful to express the yield in terms of strain, in which case we find that $\Phi = \mu \|\text{dev}[^e\varepsilon]\|_F + \alpha \frac{k}{2} (J^2 - 1)$. The Drucker-Prager yield criterion corresponds intuitively with the notion of Coulomb friction in the discrete setting; the shear stress (continuum analog of force in the tangential plane) is capped at the magnitude of the pressure (continuum analog of...
the normal force) scaled by a constant $\alpha$ (continuum analog of $\mu$). When $\Phi \leq 0$, the material behaves elastically. When $\Phi > 0$, however, the material sustains plastic shape change. Crucially, a Drucker-Prager treatment of yield allows us to tune the angle of repose of a pile (Fig. 3.5) to match that of a discrete element simulation.

We model the granular medium as perfectly plastic: all excess yield is immediately converted into plastic deformation. We discretize the elastoplastic update rule for the strain $\dot{b}^e = \nabla v b^e + b^e \nabla v^T + L_v b^e$ with the return mapping method [Simo and Hughes, 1998]. The result is a predictor-corrector style update for the strain. We first update $b^e$ through $\dot{b}^e = \nabla v b^e + b^e \nabla v^T$, ignoring any plastic flow. This predicted strain can now violate the yield condition. To remove yield excess from $b^e_*$, where * denotes some violating state, we impose two constraints on the plastic flow to project the material to a violation-free state. First, the plastic granular flow should conserve volume, and the projected strain $\det [b^e_0] = \det [b^e_*]$.

Second, to satisfy the yield condition, we seek $\Phi (\tau (b^e_0)) = 0$. We consider the flow to be in the direction of the shear and we decompose the projected strain as $b^e_0 = \lambda_1 I + \lambda_2 \det [\dev [b^e_*]]$.

Expanding the constraint $\Phi (\tau (b^e_0)) = 0$, we find:

$$
\Phi (\tau (b^e_0)) = \| \dev [\tau (b^e_0)] \|_F + \frac{\alpha}{2} \Tr [\tau (b^e_0)] \\
= \mu \| \dev [b^e_0] \|_F + \frac{\alpha \kappa}{2} (J^2 - 1) \\
= \mu \lambda_2 \| \dev [b^e_*] \|_F + \frac{\alpha \kappa}{2} (J^2 - 1)
$$

Equating to 0 and solving for $\lambda_2$, we conclude that $\lambda_2 = - \frac{\alpha \kappa}{2} (J^2 - 1) / (\mu \| \dev [b^e_*] \|_F)$. Similarly, if we expand $\det [b^e_0]$ we find (recalling that $\det [I + A] = 1 + \det [A] + \Tr [A]$):

$$
\det [b^e_0] = \det [\lambda_1 I + \lambda_2 \det [b^e_*]] \\
= \lambda_1^2 \det [I + \frac{\lambda_2}{\lambda_1} \det [b^e_*]] \\
= \lambda_1^2 \left( 1 + \frac{\lambda_2^2}{\lambda_1^2} \det [\dev [b^e_*]] \right) \\
= \lambda_1^2 + \lambda_2^2 \det [\dev [b^e_*]]
$$

Equating to $\det [b^e_*]$ and solving for $\lambda_1$ we find $\lambda_1 = \sqrt{\det [b^e_*] - \lambda_2^2 \det [\dev [b^e_*]]}$. With $\lambda_1$ and $\lambda_2$ in hand, we are able to easily project $b^e_*$ and enable the plastic flow. See Algorithm [15] for a compact description of the overall projection procedure.
3.3.3 MPM Simulation

We simulate the plastic continuum medium using the material point method \[\text{Sulsky et al., 1994}\]. Material properties are advected in a Lagrangian fashion with a spatial sampling of material points, while forces are computed on an Eulerian background grid.

We use a standard, explicit MPM integrator with a return-mapping style treatment of plasticity \[\text{Yue et al., 2015}\]. While similar works advocate for cubic-spline based nodal shape functions \[\text{Stomakhin et al., 2013; Steffen et al., 2008}\], we found these to be expensive to compute in practice. As linear hat functions are known to cause instabilities when material points cross grid boundaries \[\text{Jiang et al., 2016}\], we instead employ a version of the Generalized Interpolation Material Point Method (GIMP) \[\text{Bardenhagen and Kober, 2004}\], which, conceptually, assigns a finite width to material points to smooth the transition of the shape functions across grid boundaries. We found GIMP to be both stable and efficient. As we treat all interactions with boundaries through a discrete particle model, no special consideration is needed for boundary conditions within our material point formulation.

We summarize our MPM implementation in Algorithm 5. Note that we divide the MPM integrator into two phases for reasons that we discuss in Section 3.3.6.

Algorithm 5 MPM Step

1: MPM Step First Phase
2: MPM Step Second Phase

Algorithm 6 MPM Step First Phase

1: Rasterize Mass And Momentum To Grid ▷ Grid operation
2: Compute Stress At Points ▷ Point operation
3: Compute Forces On Grid ▷ Grid operation
4: Update Momentum On Grid ▷ Grid operation

3.3.4 Hybridization

Having accounted for the velocity updates from the equations of motion, we can interpret each system in terms of forces-per-volume and then, within a finite volume element, correct the velocities and positions to enforce coupling via \[3.1\] subject to constraint \(C\). This allows us to
Algorithm 7 Rasterize_Mass_And_Momentum_To_Grid
1: for point ∈ Material_Points do
2:     for node ∈ Stencil(point) do
3:         w ← Weight(point, node)
4:         node.m += w · point.m
5:         node.p += w · point.m · point.v
6:     end for
7: end for

Algorithm 8 Compute_Stress_At_Points
1: for point ∈ Material_Points do
2:     τ ← 0
3:     if point.J ≤ 1 then
4:         τ ← $\frac{\kappa}{2} \cdot (point.J^2 - 1) \cdot I + \mu \cdot \text{dev}[point.\bar{e}]$
5:     end if
6:     point.σ ← τ / point.J
7: end for

Algorithm 9 Compute_Forces_On_Grid
1: for point ∈ Material_Points do
2:     for node ∈ Stencil(point) do
3:         $\nabla w ← \text{Weight_Grad}(point, node)$
4:         node.f += -point.V · point.J · point.σ · $\nabla w$
5:     end for
6: end for
7: for node ∈ Grid_Nodes do
8:     node.f += node.m · g
9: end for

Algorithm 10 Update_Momentum_On_Grid
1: for node ∈ Grid_Nodes do
2:     node.p_{new} ← node.p + dt · node.f
3: end for
Algorithm 11 MPM_STEP_SECOND_PHASE

1: Lumped_Mass_Velocity_Update_On_Grid  \( \triangleright \) Grid operation
2: Compute_Velocity_Gradient_At_Points  \( \triangleright \) Point operation
3: Elastic_Prediction_At_Points  \( \triangleright \) Point operation
4: Plastic_Correction_At_Points  \( \triangleright \) Point operation
5: Update_Velocities_At_Points  \( \triangleright \) Point operation
6: Update_Positions_At_Points  \( \triangleright \) Point operation

Algorithm 12 Lumped_Mass_Velocity_Update_On_Grid

1: \textbf{for} node \( \in \text{Grid}\_\text{Nodes} \) \textbf{do}
2: \hspace{1em} node.\textbf{v} \leftarrow \text{node.\textbf{p}}_{\text{new}} / \text{node.m}
3: \hspace{1em} node.\textbf{a} \leftarrow (\text{node.\textbf{p}}_{\text{new}} - \text{node.\textbf{p}}) / (\text{dt} \cdot \text{node.m})
4: \textbf{end for}

Algorithm 13 Compute_Velocity_Gradient_At_Points

1: \textbf{for} point \( \in \text{Material}\_\text{Points} \) \textbf{do}
2: \hspace{1em} point.\textbf{v} \leftarrow 0
3: \hspace{1em} \textbf{for} node \( \in \text{Stencil}(\text{point}) \) \textbf{do}
4: \hspace{2em} \textbf{w} \leftarrow \text{Weight_Grad(}\text{point, node})
5: \hspace{2em} point.\textbf{v} += point.\textbf{v} \cdot \text{w}^T
6: \hspace{1em} \textbf{end for}
7: \textbf{end for}

Algorithm 14 Elastic_Prediction_At_Points

1: \textbf{for} point \( \in \text{Material}\_\text{Points} \) \textbf{do}
2: \hspace{1em} \textbf{b}^e \leftarrow \text{point.}J \cdot \text{point.\textbf{b}}^e
3: \hspace{1em} \textbf{b}^* \leftarrow \textbf{b}^e + \text{dt} \cdot (\text{point.\textbf{v}} \cdot \textbf{b}^e + \textbf{b}^e \cdot \text{point.\textbf{v}}^T)
4: \hspace{1em} \text{point.}J \leftarrow \sqrt{\det(\textbf{b}^*)}
5: \hspace{1em} \text{point.\textbf{b}}^e \leftarrow \textbf{b}^* / \text{point.}J
6: \textbf{end for}
Algorithm 15 Plastic_Correction_At_Points

1: for point ∈ Material_Points do
2:      yield_threshold ← $-\alpha \cdot \frac{\kappa}{2} \cdot (\text{point}.J^2 - 1)$
3:      $\text{dev}[\bar{\mathbf{b}}^e] ← \text{point}.\mathbf{b}^e - \frac{1}{2} \cdot \text{Tr} [\text{point}.\mathbf{b}^e] \cdot \mathbf{I}$
4:      $\text{dev}[\bar{\mathbf{b}}^e] ← \text{dev}[\mathbf{b}^e] / \text{point}.J$
5:      if $\mu \cdot \| \text{dev}[\bar{\mathbf{b}}^e] \|_F > \text{yield_threshold}$ then
6:         $\lambda_2 ← \text{yield_threshold} / (\mu \cdot \| \text{dev}[\bar{\mathbf{b}}^e] \|_F)$
7:         $\lambda_1 ← \sqrt{\det [\text{point}.\mathbf{b}^e] - \lambda_2^2 \cdot \det [\text{dev}[\mathbf{b}^e]]}$
8:         $\text{point}.\mathbf{b}^e ← \lambda_1 \cdot \mathbf{I} + \lambda_2 \cdot \text{dev}[\mathbf{b}^e]$
9:      end if
10: end for

Algorithm 16 Update_Velocities_At_Points

1: for point ∈ Material_Points do
2:      $\mathbf{v}_{\text{pic}} ← 0$
3:      $\mathbf{a}_{\text{flip}} ← 0$
4: for node ∈ Stencil(point) do
5:      $w ← \text{Weight}(\text{point}, \text{node})$
6:      $\mathbf{v}_{\text{pic}} += w \cdot \text{node}.\mathbf{v}$
7:      $\mathbf{a}_{\text{flip}} += w \cdot \text{node}.\mathbf{a}$
8: end for
9: $\mathbf{v}_{\text{flip}} ← \text{point}.\mathbf{v} + dt \cdot \mathbf{a}_{\text{flip}}$
10: $\text{point}.\mathbf{v} ← (1 - \alpha) \cdot \mathbf{v}_{\text{pic}} + \alpha \cdot \mathbf{v}_{\text{flip}}$
11: end for
Algorithm 17 Update

\textbf{Algorithm 17 Update\_Positions\_At\_Points}

1: \textbf{for} point $\in$ Material\_Points \textbf{do}
2: \hfill $v_{pic} \leftarrow 0$
3: \textbf{for} node $\in$ Stencil(point) \textbf{do}
4: \hfill $w \leftarrow \text{Weight}(\text{point}, \text{node})$
5: \hfill $v_{pic} += w \cdot \text{node}.v$
6: \textbf{end for}
7: \hfill \text{point}.x += dt \cdot v_{pic}$
8: \textbf{end for}$

interpret the constraint $v_1 = v_2$ in an average or homogenized sense \cite{Bergou et al., 2007}: in the reconciliation zone, the average velocity of the discrete particles within a localized element agrees with that of the continuum. Let $\lambda_k$ represent the constraint force on the $k$th discrete particle. Given the reconciliation zone, $\Omega_R$, the $p$th material point moves as

$$\frac{d}{dt} q_p = v_p$$

$$\frac{d}{dt} (w_p M_p v_p) = \frac{d}{dt} (w_p M_p v_p^*) + \sum_{k \in \Omega_R} \Gamma_{pk} \lambda_k$$

while the $k$th discrete particle moves as

$$\frac{d}{dt} q_k = v_k$$

$$\frac{d}{dt} ((1 - w_k) M_k v_k) = \frac{d}{dt} ((1 - w_k) M_k v_k^*) + \lambda_k$$

where $v_p^*$ ($v_k^*$) are the predictions from the continuum (discrete) simulation before coupling forces are added, and $\Gamma_{pk}$ are material-point to discrete-particle interpolation coefficients.
3.3.5 Derivation of Predictor-Corrector Hybrid Time Integrator

Consider now the generic (discretized) systems $G_c$ and $G_d$ coupled through constraints $C$. The state at the next time-step (denoted by the 1 superscript) is given by parameters that satisfy:

$$
G_c(v^1_c, \lambda^1) = 0 \\
G_d(v^1_d, \lambda^1) = 0 \\
C(v^1_c, v^1_d) = 0
$$

Note that while the $G$ systems can be explicit or implicit, we presume that the constraints are implicit and thus evaluated at the end-of-step state. We omit $x$ for clarity, noting that for typical integrators of interest $x$ can be expressed in terms of $v$.

A generic Newton-like algorithm for time-stepping this system takes the form of Alg. 18.

<table>
<thead>
<tr>
<th>Algorithm 18 Coupled Step $(v^0_c, v^0_d)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1: $(v_{c,0}, v_{d,0}, \lambda_0) \leftarrow \text{Generate Initial Iterate}$</td>
</tr>
<tr>
<td>2: for $k = 0 \ldots \text{Max Iterations}$ do</td>
</tr>
<tr>
<td>3: $G_{c,k} \leftarrow G_c(v_{c,k}, \lambda_k)$ \hspace{1cm} \triangleright RHS</td>
</tr>
<tr>
<td>4: $G_{d,k} \leftarrow G_d(v_{d,k}, \lambda_k)$</td>
</tr>
<tr>
<td>5: $C_k \leftarrow C(v_{c,k}, v_{d,k})$</td>
</tr>
<tr>
<td>6: Residual $\leftarrow</td>
</tr>
<tr>
<td>7: if Residual $\leq \epsilon$ then</td>
</tr>
<tr>
<td>8: Break</td>
</tr>
<tr>
<td>9: end if</td>
</tr>
<tr>
<td>10: $\frac{\partial G_c}{\partial v^1_c k} \leftarrow \frac{\partial G_{c,k}}{\partial v^1_c (v_{c,k})}$ \hspace{1cm} \triangleright LHS</td>
</tr>
<tr>
<td>11: $\frac{\partial G_d}{\partial v^1_d k} \leftarrow \frac{\partial G_{d,k}}{\partial v^1_d (v_{d,k})}$</td>
</tr>
<tr>
<td>12: $(\Delta v_c, \Delta v_d, \Delta \lambda) \leftarrow \text{Solve} \left( G_{c,k}, G_{d,k}, C_k, \frac{\partial G_c}{\partial v^1_c k}, \frac{\partial G_d}{\partial v^1_d k} \right)$</td>
</tr>
<tr>
<td>13: $v_{c,k+1} \leftarrow v_{c,k} + \Delta v_c$</td>
</tr>
<tr>
<td>14: $v_{d,k+1} \leftarrow v_{d,k} + \Delta v_d$</td>
</tr>
<tr>
<td>15: $\lambda_{k+1} \leftarrow \lambda_k + \Delta \lambda$</td>
</tr>
<tr>
<td>16: end for</td>
</tr>
</tbody>
</table>
The linear system in line 12 of this Newton solve is given by:
\[
\begin{bmatrix}
  \frac{\partial G_c}{\partial v_c^1} & 0 & \Gamma_c \\
  0 & \frac{\partial G_d}{\partial v_d^1} & -\Gamma_d \\
  \Gamma_c^T & -\Gamma_d^T & 0
\end{bmatrix}
\begin{bmatrix}
  \Delta v_c \\
  \Delta v_d \\
  \Delta \lambda
\end{bmatrix}
= \begin{bmatrix}
  -G_c \\
  -G_d \\
  -C
\end{bmatrix}
\]

To arrive at our hybrid integrator, first note that if both integrators are explicit, the non-linear solve will reduce to a single linear solve, with \( \Delta v_c = v_c^1 - v_c^0 \) and \( \Delta v_d = v_d^1 - v_d^0 \). Next, observe that if both integrators are explicit, \( \frac{\partial G_c}{\partial v_c^1} = M_c \) and \( \frac{\partial G_d}{\partial v_d^1} = M_d \). Finally, we have the choice of initial iterate in Alg. If we set \( v_{c,0} = 0, v_{d,0} = 0, \) and \( \lambda_0 = 0 \), the linear system reduces to that of (3.3), revealing the predictor-corrector structure of the method.

### 3.3.6 Discrete Hybrid Coupling

We now present the discretized, hybrid coupling algorithm that follows from the previous general discussion. We begin by noting that MPM forces are defined on a background Eulerian grid, which will provide a natural scratch-pad on which to compute constraint forces. As such, we begin a time-step with a full predictor step of the discrete system, and a half-step of the continuum system to the instant that material point properties are rasterized to the grid. We next assemble and solve a linear system to constrain the velocities of the discrete and continuum simulations in the reconciliation zone
\[
\begin{bmatrix}
  W_c M_c & 0 & \Gamma_c \\
  0 & W_d M_d & -\Gamma_d \\
  \Gamma_c^T & -\Gamma_d^T & 0
\end{bmatrix}
\begin{bmatrix}
  v_c^{n+1} \\
  v_d^{n+1} \\
  \lambda
\end{bmatrix}
= \begin{bmatrix}
  W_c (M_c v_c^n + h f_c^n) \\
  W_d (M_d v_d^n + h f_d^n) \\
  0
\end{bmatrix}
\]

where \( W_c \) and \( W_d \) are diagonal matrices that contain the mass weights for the continuum and discrete systems, \( M_c \) and \( M_d \) are the mass matrices, \( v_c \) and \( v_d \) are the velocities, and \( f_c \) and \( f_d \) are explicit forces from the continuum and discrete systems. \( \Gamma_d \) and \( \Gamma_c \) are defined such that \( \Gamma_d^T v_d - \Gamma_c^T v_c \) produces the residual relative velocity of discrete bodies within the background velocity field defined by the material point grid. Under this definition, \( \Gamma_d \) reduces to the identity matrix, while each column of \( \Gamma_c \) contains the weights that recover each discrete body’s center of mass from the MPM grid’s basis functions. Note that we can restrict the size of this system to only the degrees of freedom in the reconciliation zone. After solving the linear system, we
update the discrete velocities and update the positions of hybrid bodies according to their new velocities, concluding the discrete step, and we compute the second half of the material point step using the new, constrained velocities, concluding the continuum step. This concludes a full hybrid time-step, the details of which are summarized in Alg. 19.

While this method to compute coupling forces indeed works, a further speedup is possible by defining a second background grid that is co-located with the background MPM grid. The velocities of the discrete bodies can be represented on this second grid as if they were material points. The constraint matrices $\Gamma_d$ and $\Gamma_c$ now reduce to the identity, and the system in (3.3) can be solved in closed form. When constraining the velocities of co-located grids to match, we arrive at the following system of equations:

$$W_c M_c v_c^{n+1} + \lambda = W_c M_c v_c^*$$  \hspace{1cm} (3.4)

$$W_d M_d v_d^{n+1} - \lambda = W_d M_d v_d^*$$  \hspace{1cm} (3.5)

$$v_c^{n+1} = v_d^{n+1}$$  \hspace{1cm} (3.6)

Substituting $v_c^{n+1}$ for $v_d^{n+1}$ in Eq. (3.4), adding Eq. (3.4) and Eq. (3.5), and solving for $v_d^{n+1}$, we find that:

$$v_c^{n+1} = v_d^{n+1} = (W_c M_c + W_d M_d)^{-1} (W_c M_c v_c^* + W_d M_d v_d^*)$$

For diagonal mass matrices, each degree of freedom can be solved for independently, and the formula reduces to an inelastic impact between two particles in one dimension. This method to compute hybridization constraint forces is simple, robust, and trivially parallelized. After solving this system, the discrete grid-based velocities are mapped back to the discrete bodies in the same manner as MPM points.

### 3.4 Hybrid Oracle

With a coupling method in place, a full hybrid granular simulation method requires three additional core features: an ‘oracle’ to identify regions that can be safely approximated as a continuum, a homogenization operator that is able to convert a discrete region of material to a continuum region, and an enrichment operator that is able to synthesize discrete grains in a manner consistent with the continuum.
CHAPTER 3. HYBRID SIMULATION OF GRANULAR MATERIALS

Algorithm 19 Hybrid Step
1: MPM Step First Phase
2: Discrete Step
3: Rasterize Discrete And Continuum Velocities
4: Hybrid Constraint Solve
5: Transfer Constrained Velocities To Discrete And Continuum
6: MPM Step Second Phase
7: Update Hybrid Discrete Positions
8: Update Hybrid Regions And Resample

Algorithm 20 Hybrid Constraint Solve
1: for $i = 0 \ldots \text{num}_\text{grid}_\text{nodes}$ do
2: \[ \mathbf{p}_i \leftarrow w_{c,i} \cdot m_{c,i} \cdot \mathbf{v}_{c,i} + w_{d,i} \cdot m_{d,i} \cdot \mathbf{v}_{d,i} \] \hspace{1em} \triangleright Colocated grid momentum
3: \[ m_i \leftarrow w_{c,i} \cdot m_{c,i} + w_{d,i} \cdot m_{d,i} \] \hspace{1em} \triangleright Colocated grid mass
4: if $m_i \neq 0$ then
5: \[ \mathbf{v}_i \leftarrow \mathbf{p}_i / m_i \] \hspace{1em} \triangleright Constrained grid velocity
6: else
7: \[ \mathbf{v}_i \leftarrow 0 \]
8: end if
9: end for
Figure 3.6: Initialization of a hybrid simulation: (A) We begin with a collection of discrete grains. (B) We next locate a level set corresponding to a given low density, here denoted as a black line. (C) Across the simulation domain, we compute the distance to the density threshold, here indicated by lines in lighter shades of red as the distance increases. (D) We select a user-tunable distance to the density level-set that will serve as the center of the hybrid ‘reconciliation’ zone. We indicate this critical distance here as a solid black line. (E) We extend the hybrid zone along the distance field by a given width in each direction, indicated here by dotted lines. This hybrid reconciliation zone between the dotted lines defines a zone where the discrete system will be coupled to the continuum system. (F) We delete all discrete grains that fall within the inner boundary of the reconciliation zone. (G) We run the ‘avoid-a-void’ algorithm of Yue et al. [2015] from the outer boundary in to populate the region with material points.
3.4.1 A Continuum Oracle

Critical to our hybridization method is an oracle that is able to flag regions of the simulation domain as safe for a continuum treatment. Regions are unfit for a continuum treatment when one of any number of conditions are satisfied. First, in regions of low pressure, grains are more likely to separate from the material bulk and undergo ballistic motion. Second, high strain rate gradients suggest that the granular flow varies too rapidly to be safely represented as a homogenized continuum [Dijksman and van Hecke, 2010; Kamrin, 2010b; Koval et al., 2009]. Finally, in thin flows, grain-level dynamics can dominate, leading to finite size effects (e.g., jamming) not captured by local continuum models [Beverloo et al., 1961; MiDi, 2004; Pouliquen, 1999; Sheldon and Durian, 2010]. We have found that the packing fraction serves as an effective stand-in for these sources of fundamentally discrete behavior. Low pressures occur near free surfaces, which will contain looser packings, rapid fluctuations in granular flows are more likely with looser packings, and thin features naturally correlate with lower densities.

Our oracle begins by computing the packing fraction of the discrete particle system on a uniform, background grid. Note that only discrete grains are considered when computing the packing fraction, as continuum and hybrid regions are, by ansatz, considered sufficiently dense. From this implicit representation, we extract an isocontour corresponding to a critical, threshold packing fraction. We next compute the distance to this threshold isocontour on a second, uniform grid. From the distance to the threshold, we partition space into three disjoint regions: a fully continuum zone whose distance is less than a given hybrid zone half-width, a fully discrete zone whose distance is greater than the given hybrid zone half-width, and a hybrid reconciliation zone whose distance lies within the half-width (Alg. 21). See Figure 3.6 for an example of initializing a hybrid simulation from purely discrete initial conditions.

3.4.2 Homogenization and Enrichment

After updating the boundary between simulation domains, we are faced with four possible transition scenarios: a previously hybrid zone is now purely discrete, a previously hybrid zone is now purely continuum, a previously discrete zone is now hybrid, or a previously continuum zone is now hybrid. Note that after initialization, we do not permit direct transitions from continuum to discrete regions or vice versa. The transition away from a hybrid representation
is quite simple. For a hybrid region transitioning to a purely discrete region, we simply delete all material points in the region. Similarly, for a hybrid region transitioning to a purely continuum region, we delete all discrete grains in the region. The transition to a hybrid representation is more involved, however. A region that previously contained only material points will require the insertion of discrete grains. Likewise, a previously discrete region will require the insertion of new material points. See Alg. 22

The problem of adding samples to a dynamic simulation was addressed in the context of the material point method with the recently proposed avoid-a-void algorithm \cite{Yue et al., 2015}. The avoid-a-void method applies Poisson disc sampling to maintain approximately constant material point distributions and to prevent the formation of non-physical voids within a simulated material. This technique is perfectly suited to our needs, where we need to insert material points or discrete grains in regions of high material density recently labeled as hybrid. For discrete grains, the new position is determined by the Poisson disc sampling procedure, while we draw the radius of new grains from the same normal distribution used to generate the initial grain radii. The initial velocity of new discrete grains is computed by averaging the velocity of surrounding discrete grains within a radius of 6 (mean) grain diameters, with an exponential falloff (Alg. 24). This window width is slightly above the minimal size that recovers continuum-like quantities in a ‘granular volume element’ \cite{Rycroft et al., 2009}. These nearest neighbor queries are accelerated with a uniform, background grid. New material points are generated in the same fashion. The positions of new material points are computed with Poisson disc sampling, while the initial velocity, (normalized) strain, and deformation gradient magnitude are computed by averaging the quantities from neighboring material points (Alg. 25).

\begin{algorithm}
\caption{Identify_Hybrid_Zones}
1: $\Phi_p \leftarrow \text{Discrete\_Packing\_Fraction\_Isocontours}(q_d)$
2: $\Phi_d \leftarrow \text{Distance\_to\_Density\_Isocontours}(\Phi_p)$
3: $\text{continuum\_zone}(x) \leftarrow \Phi_d(x) > \phi_0 - r_h$
4: $\text{discrete\_zone}(x) \leftarrow \Phi_d(x) < \phi_0 + r_h$
5: $\text{hybrid\_zone}(x) \leftarrow \phi_0 - r_h \leq \Phi_d(x) \leq \phi_0 + r_h$
\end{algorithm}
Algorithm 22 Update Hybrid Zones
1: Discrete Avoid a Void (hybrid zone)
2: Continuum Avoid a Void (hybrid zone)
3: Delete Discrete Grains (continuum zone)
4: Delete Continuum Particles (discrete zone)

Algorithm 23 Update Hybrid State
1: Identify Hybrid Zones
2: Update Hybrid Zones

Algorithm 24 Create Discrete Grain
1: $x \leftarrow \text{Position from Avoid a Void}$
2: $r \leftarrow N(r_{\text{mean}}, r_{\text{sigma}})$  \hspace{1cm} \triangleright \text{Normally distributed radii}$
3: $m \leftarrow \frac{4}{3} \pi r^3$
4: $v \leftarrow 0$
5: $W \leftarrow 0$
6: for $i = 0 \ldots N_d$ do
7: \hspace{0.5cm} if $|x_i - x| < 12 r_{\text{mean}}$ then
8: \hspace{1cm} $w \leftarrow e^{[|x_i - x|^2/2r^2_{\text{mean}}}$
9: \hspace{1cm} $v \leftarrow v + w \cdot v_i$
10: \hspace{1cm} $W \leftarrow W + w$
11: \hspace{0.5cm} end if
12: end for
13: $v \leftarrow v/W$
Algorithm 25 Create_Material_Point

1: \( x \leftarrow \text{Position\_from\_Avoid\_a\_Void} \)

2: \( r \leftarrow r_{MPM} \) \hfill \text{\( r_{MPM} \) \ Constant material point radius}

3: \( m \leftarrow m_{MPM} \) \hfill \text{\( m_{MPM} \) \ Constant material point mass}

4: \( v \leftarrow 0 \)

5: \( J \leftarrow 0 \)

6: \( \bar{b} \leftarrow 0 \)

7: \( W \leftarrow 0 \)

8: \textbf{for} \( i = 0 \ldots N_c \) \textbf{do}

9: \quad \textbf{if} \ |x_i - x| < 12 r_{MPM} \textbf{ then}

10: \quad \quad w \leftarrow e^{\frac{|x_i - x|}{2r_{MPM}}^2}

11: \quad \quad v \leftarrow v + w \ v_i

12: \quad \quad J \leftarrow J + w \ J_i

13: \quad \quad \bar{b} \leftarrow J + w \ \bar{b}_i

14: \quad \quad W \leftarrow W + w

15: \quad \textbf{end if}

16: \textbf{end for}

17: v \leftarrow v / W

18: J \leftarrow J / W

19: \bar{b} \leftarrow \bar{b} / W
CHAPTER 3. HYBRID SIMULATION OF GRANULAR MATERIALS

3.5 Results

We now test our method against two model problems: a column collapse, and grain discharge from a silo. In each of these examples, we employ uniform density partition weights of $\frac{1}{2}$.

**Granular Column Collapse.** In Figure 3.7 we simulate a collapsing column of grains with both a purely discrete method and with our hybrid approach. Note the correspondence between the shapes of both piles. Further observe that our hybrid method is able to capture detailed ‘fly away’ effects – individual grains are allowed to separate from the overall bulk and roll away at the front of the collapse, a visually important effect that would be difficult to capture with a purely continuum model.

Encouraged by the agreement between the purely discrete approach and our hybrid approach, we plan to validate our hybrid model against the power-law scaling of the run-out distance reported in the literature. Granular run-out in a column has been shown to follow a power law scaling as a function of the initial aspect ratio in both experimental [Lube et al., 2005] and numerical [Staron and Hinch, 2005; Lagrée et al., 2011] tests. Running a series of run-out simulations over a range of aspect ratios, we will attempt to corroborate the previously reported power law scaling. Below a critical aspect ratio, we should observe a linear run-out distance as a function of aspect ratio. Above this threshold, we should observe a second power law scaling.

**Silo discharge.** In Figure 3.8 we simulate a silo discharging grains using a purely discrete approach and with our hybrid approach. With our approach, the oracle initially identifies the
Our Hybrid Simulation

Discrete
Continuum
grains
regime

Discrete Element Simulation
Oracle
initiates
homogen-
ization

$t = 0$
$\phi_{33}$
$t = 0$
$\phi_{67}$
$t = \infty$
$\phi_{33}$
$t = 0$
$t = 0$
$\phi_{33}$
$t = 0$
$\phi_{67}$
$t = \infty$
$\phi_{33}$

Figure 3.8: Silo discharge: A silo that is discharging grains is simulated with a discrete method (left), and with our hybrid method (right).

interior of the initial mass of grains as a continuum. As grains exit the silo and the continuum region falls towards the orifice, our method automatically converts the continuum material to discrete material. As grains form a pile on the ground, our method detects the formation of the sufficiently dense portions of the pile and automatically converts discrete grains to continuum material points in this area.

An advantage of our hybrid approach over a purely continuum method is the ability of our simulations to frictionally jam due to so-called finite size effects. We are able to capture jamming effects with our hybrid model. In Figure 3.9, we simulate a silo discharge with a large orifice width and with a small orifice width using our hybrid algorithm. Our hybrid simulation jams with the small orifice width, as expected, but permits flow with the larger orifice width.

In the near future we plan to explore whether our hybrid method can reproduce the findings of Beverloo [Beverloo et al., 1961], who identified a scaling law for the flow rate from a funnel as a function of orifice size.

Penetrometer insertion. Similar to Yan [Yan et al., 2010] and Wellmann and Wriggers, 2012, we perform a hybrid simulation of a penetrometer insertion into a bed of grains (Fig. 3.10). These simulations are difficult to perform directly with standard continuum methods owing to the massive plastic shape changes observed around the penetrometer tip. Unlike previous works, we do not need to specify the region to be treated with a discrete
Figure 3.9: Frictional jamming: A silo discharge simulated with our hybrid algorithm for a large orifice width (left) and for a small orifice width (right). For the large width, grains are able to freely flow from the funnel as the oracle enriches the region near the orifice. For the small width, the granular assembly jams after a tenth of a second, halting all flow from the orifice.

Figure 3.10: Penetrometer insertion: We insert a penetrometer into a bed of grains with our hybrid algorithm (first four frames). As the penetrometer enters the bed, our hybrid oracle identifies the region around the tip as requiring a discrete treatment and enriches the simulation domain in this area. As the simulation progresses, the continuum region eventually experiences a topology change and splits in two. Examining an overlay of a purely discrete simulation on top of a hybrid simulation, we find the resulting profiles to be in almost perfect agreement (rightmost frame).
Figure 3.11: Spinning drum: We rotate a drum filled halfway with granules with a discrete element method (top row) and with our hybrid algorithm (bottom row). As the system evolves, observe that the shape of the free surface obtained with our hybrid method agrees with that of the purely discrete method.

Instead, as the penetrometer advances into the bed of grains, our hybrid method is able to enrich the region surrounding the penetrometer, ensuring that it always interacts with the bed through discrete grains. As the penetrometer is fully inserted into the bed, the original single continuum region is split in two. Our hybrid approach gracefully treats this topological change with no extra machinery.

**Spinning drum.** An understanding of drum-like geometries is important in both industrial applications (e.g. mills, tumblers) and in the study of free-surface granular flows [G.D.R. MiDi, 2004]. To assess whether our algorithm is suitable for these drum geometries, we fill a drum with grains to half its area, and impose a rotation to the drum with a constant angular velocity. With a discrete element simulation, we observe nearly rigid grains near the base of the drum, a steadily increasing flow towards the interior of the granular assembly, and loosely packed grains near the free surface. As the transient phase subsides, we observe the characteristic free-surface shape of these experiments. Comparing the purely discrete results to those from our hybrid algorithm (Fig. 3.11), we find the profiles to be in good agreement throughout the evolution of the simulation. Because our hybrid algorithm treats regions near surfaces with discrete grains,
we do not require any additional machinery to handle the drum boundary condition beyond that from the discrete simulation. Like the discrete simulation, our hybrid algorithm is also able to capture free flight flyaway grains near the top of the domain.

3.6 Conclusion and Future Work

We have presented a theoretical framework for coupling a continuum model of a granular material to a discrete model. To develop this framework, we have proposed a method to partition a granular material into two distinct but overlapping systems. By modeling one system with a discrete model, and one system with a continuum model, we arrive at a hybrid treatment for a granular material. We then presented a discrete version of this framework able to accommodate a variety of integrators, and derived an explicit hybrid integrator that couples a Material Point simulation of a continuum to a discrete element simulation. We have demonstrated preliminary hybrid simulations with this integrator in settings that experience both large plastic shape changes and topology changes. In the immediate term, we are actively validating our method by exploring whether the approach can reproduce both Beverloo scaling in a funnel discharge and the correct run-out distances in a collapsing column of grains.

Longer term, with our approach validated in 2D, we seek validate our preliminary 3D integrator (Figure 3.12) on lab-scale experiments, laying the foundation for a powerful tool to study granular materials. Beyond validation, we are interested in extending our approach to treat both systems implicitly. The coupling framework naturally extends to other integrators than those we have chosen, and a fully implicit approach promises to yield significant performance improvements when integrating stiff systems, including granular materials. As contact dynamics models are implicit, an extension to implicit integrators would allow us to treat these models as well. Improvements are possible to our approach by constraining the angular momentum of the discrete system: if a grain with non-zero angular momentum and no active contacts is embedded in the hybrid zone, it will spin in place until it comes back into contact with the discrete system. While there is no notion of point-wise angular momentum in a continuum, certain discretizations, including the Affine Particle in Cell method, track angular momentum-like quantities, to which we could constrain the angular momentum. We are also interested in refin-
Figure 3.12: 3D column collapse: A cutaway rendering of a column collapse computed using our preliminary 3D hybrid algorithm. Material points are hidden to reveal the region treated with a continuum.

In addition to the mapping and exploring the limits of our methodology for detecting regions safe for homogenization and our methodology for resampling regions as the boundaries between continuum and discrete evolve. Finally, while we currently use uniform density weights of $\frac{1}{2}$, we are interested in exploring how the use of smoothly varying weights might change the simulation results. In particular, our use of non-smooth weights could cause artifacts when analyzing wave propagation in granular systems. Smoothly varying weights could be defined by solving a Laplacian in the hybrid reconciliation zone (simulating a diffusion, in effect).
Chapter 4

Automated Verification of Numerical Simulations for Computer Science Instruction

Physically based computer animation is a field of research that uses the laws of physics to automatically synthesize novel animations. Physically based animation drives technology used across the film industry, including tools to simulate environmental effects like the motion of fluids and flames, as well as tools to simulate features of animated characters, like clothing, hair, skin, and flesh. These tools allow artists to generate detail with a degree of fidelity that would otherwise be impractical or impossible to animate by hand. In video games and related applications, software must respond to unpredictable and wildly varying user inputs; physically based animation tools generate plausible responses to these user inputs, even with strict computational budgets [Coumans, 2016, Bender et al., 2014b]. Within the creative software industry, physically based animation enables compelling paint simulations, allowing artists to paint with water- [Chu and Tai, 2005] and oil-based [Baxter et al., 2004] paints using realistically simulated brushes [DiVerdi et al., 2010]. Physics is often used to drive animations in user interfaces, providing touch based mobile interfaces with a feeling of connection to the real world [Apple, 2018]. Web-based data visualization frameworks even incorporate physically based animation techniques, providing new and intuitive ways to manipulate data [Bostock, 2011].
Finally, bringing a unique, computer science centric view to the simulation of physical phenomena, these techniques have cross-disciplinary impact, with applications even appearing in surgical training [Chentanez et al., 2009].

With the proliferation of physically based animation tools, we seek to equip students with the background necessary to develop, debug, and work with these techniques. Topics that fall in the domain of physically based animation draw from a wide range of scientific and mathematical disciplines, introducing a necessarily extensive breadth to the course material. At the same time, we seek to give students enough depth to understand papers published in leading venues for physically based animation, including the annual ACM SIGGRAPH conference. We also seek to encourage students to explore the creative and artistic avenues opened by technological advances in physics based animation. These competing goals – breadth of coverage, depth of understanding, and our desire to encourage creative expression – necessitate an intensive course of study over the duration of a semester. The intense pace of the course material, combined with the fact that our course is, for many students, their first introduction to the numerical heavy flavor of algorithms employed in physically based animation, somewhat limited the appeal of initial course offerings. Our goal then is to formulate a course that maintains widespread appeal and accessibility to students from a computer science background, while at the same time minimizing sacrifices to the technical breadth and depth of the curriculum.

Our methodology for teaching physically based animation draws inspiration from domains like test-driven development, continuous-integration testing, and agile software development, all of which foster a fast and iterative feedback cycle and allow for a brisk yet flexible development cadence. These methodologies are more difficult to apply to the continuous and numerical flavor of physically based animation tools and algorithms, however, than to the discrete world in which computer science students are used to working. To that end, we build our curriculum around the concept of a numerical autograder. This numerical autograder enables instant and concrete test-driven feedback with numerical code, leading to a fast paced and engaging course structure and a concrete and transparent grading framework. Finally, to encourage students to exercise not just their technical prowess but also their creative sides, we augment our core autograded assignments with both a technical competition and weekly, free-form creative assignments.

This new course structure – a numerical autograder with frequent creative assignments –
has proven successful at Columbia University. We have significantly increased student retention rates while at the same time expanding the scope of our course curriculum. Students are able to produce impressive technical and creative projects over the duration of a semester, and are well equipped to further delve into the scientific literature and to apply their new-found skills in real-world, industrial settings. This course material has been deployed as a massive open online course (MOOC) on the edX platform, which has over 14,000 active students as of February 2018. Without the autograder as an enabling technology, deployment of our course material at such a large scale would not be feasible.

4.1 The Numerical Autograder

We approach the design of a course in physically based animation with a few concrete desiderata:

1. Fast iteration through diverse technical topics;

2. Transparent and predictable feedback and grading; and

3. Implementation and testing tools that mirror how Computer Science students are accustomed to developing software.

Starting from our first goal of rapid iteration through extensive course materials, we divide our overall curriculum into five distinct themes (Section 4.2). Each theme is further subdivided into milestones with weekly deliverables in the form of self-contained coding assignments (Section 4.2). Moving to a weekly code delivery schedule in lieu of a few monolithic assignments allows us to cover a wider breadth of topics, but risks setting an unreasonable pace for students and course instructors alike. To aid students in completing assignments at a timely pace, to facilitate scalable grading, and to remove ambiguity with regards to final achieved grades, we draw inspiration from test driven development and formulate each coding assignment as a series of test simulations that a student must pass. Each milestone is presented as a document that incrementally introduces new, bite size topics, often including additional mathematical derivations required on the student’s part, with an associated feature that the student must implement in a provided starter code base. These feature requirements come with a small self-contained test scene, where students are provided with access to an auto-grading ‘oracle’ to verify the
correctness of these test scenes. The test cases that we provide for each milestone account for half of the student’s overall grade, leaving students with a sense of confidence regarding how well they will perform when we run their submission against the full test suite to determine their final grade. The development of an auto-grading oracle for physically based simulation coursework presents unique challenges, however, and leads to our core contribution.

Mathematical and Physical Underpinnings. Before discussing the details of our proposed auto-grader, we must first briefly discuss differential equations – a mathematical construct used throughout physically based animation – and certain issues that arise when attempting to computationally solve differential equations. Differential equations define relationships between the values that a function can take with the values of its derivatives. In physical settings, differential equations are typically defined over time and space, with solutions given by functions that satisfy the differential equation at all points in time and space. Examples of differential equations common in physically based animation include Newton’s equations, $F = ma$, which serve as the foundation for describing all classical mechanical systems, and the Navier-Stokes equations, which are fundamental in describing the mechanics of a fluid. With suitable conditions imposed on the initial state and the boundaries of these systems, these laws describe the system’s subsequent evolution. As such, differential equations are the fundamental tool of physically based animation, where our goal is to automatically synthesize realistic motions and animations. Closed form, analytical solutions are available for simple differential equations, but for most systems of visual interest, analytical solutions do not exist. Instead, we are forced to discretize the differential equation of interest and march the approximate solution forward in discrete time-steps. The use of discretized differential equations leads to the first difficulty in formulating an auto-grader: techniques for verifying the correctness of discrete integrators do not typically lend themselves to interactive feedback. Convergence studies involve examining the behavior of a given integrator with progressively finer levels of discretization, leading to a computational cost that prohibits instant feedback and interactive debugging. Alternatively, one can compare the discrete solution to a differential equation against a known subset of analytically-given solutions. Comparisons to analytical solutions, however, do not imply a correct implementation for all possible input conditions. More importantly, analytical solutions do not provide a means for verifying a discrete simulation’s behavior for arbitrary, general inputs.
Numerical Verification. There is a third potential solution to the verification problem, however. Given a known, vetted implementation of a given numerical technique, one can run this trusted ‘oracle’ implementation against the unknown implementation, and compare the solutions. While this solution requires one to already possess a trusted oracle, it has the advantage of enabling interactive debugging sessions with just a constant order performance overhead. The oracle solution also has the advantage of being quite general, enabling one to test any given set of inputs. While comparing against the oracle solution for a finite number of inputs does not prove the correctness of an unknown implementation, it can concretely identify *incorrect* implementations, and exploring a progressively larger, representative swath of the input problem space gives one progressively more statistical confidence in the correctness of the implementation. Finally, given an oracle capable of generating a known, verified, and reproducible solution, the delta between this oracle’s solution and the solution from the implementation under study gives one a concrete means for providing feedback on precisely where an unknown implementation is failing. These two characteristics – generality and feedback – make the oracle our tool of choice in verifying student implementations. The generality of this technique allows it to apply to all input situations, covering the breadth of material in the course, while also leading immediately to a simple and transparent grading criterion: an implementation’s performance is judged solely on its performance for a given, exhaustive set of test examples. The final score for an implementation on an assignment is given by the percentage of test inputs that it passes. This grading technique is incredibly transparent, fully automated, and leaves no room for interpretation or second guessing. By providing students with a subset of the test case collection for each assignment, we also provide students with a reasonable means for gauging their understanding on a given assignment. Finally, this oracle technique provides enough information for us to build visual debugging tools into the assignments to aid students in identifying and rectifying errors in their code. On the surface, the oracle would seem to be the perfect fit given our desiderata, but there is one major wrinkle: floating point arithmetic.

Floating Point Arithmetic. Floating point arithmetic, including the IEEE 754 variant implemented on most modern processors, is on the surface a sensible abstraction of real numbered arithmetic that belies a host of underlying subtleties. Floating point representations are a sparse sampling of the real number line: there are uncountably many reals that
Floating point representations must sample with a finite number of bits. While this sparse sampling alone can lead to confusion – simple numbers like 1.1 are not exactly representable under most floating point standards – more troublesome is the fact that fundamental properties of real numbers can fail to hold under floating point arithmetic. Floating point addition is commutative: $x + y = y + x$ holds. Floating point addition is *not* associative, however, and $(x + y) + z \neq x + (y + z)$, necessarily. This simple observation implies that two entirely valid implementations of a numerical algorithm can give slightly different results when executed with floating point operations. Compounded over multiple iterations, two completely acceptable implementations of an algorithm can yield increasingly diverging solutions (Fig 4.1). The ability of two acceptable answers to drift away from one another makes direct, cross validation of solutions tricky. These problems are exacerbated if the system is badly conditioned, in which
case small variations in the solution at one time-step can yield massively divergent trajectories in later time-steps (Fig 4.2). To this end, we propose the core component of our grading framework, the numerical auto-grading oracle.

Figure 4.2: Simulation sensitivity. An $\epsilon$-sized delta in state in a single time-step can lead to a massive global divergence in a system’s trajectory. As a tangible example, we simulate a circular ball colliding against a fixed box. On the left we impose an $\epsilon$-sized clockwise rotation of the box, while on the right we impose an $\epsilon$-sized counterclockwise rotation of the box. While these changes lead to small changes in a single time-step, they produce large differences in the global trajectory, complicating a direct comparison of the simulations’ output.

**The Numerical Oracle.** As a precondition to a numerical auto-grader, we require a trusted, vetted implementation of the algorithm of interest. Trust in this implementation can be established by expert review, through off-line validations and convergence studies, and through careful auditing of the implementation. With an oracle implementation in hand, we avoid the floating point drift described earlier by running the oracle implementation and the student implementation in tandem. At the beginning of each discrete instant of time – recall that differential equations typically treat continuously varying time by chopping that axis into discrete chunks – we seed the oracle with the state of the student’s simulation. Starting from
these identical states, we then evolve the system forward with the oracle and with the student implementation. We compare the end-of-step state of the system from the oracle with the end-of-step state from the student implementation and take note of any differences. We can then repeat this seed-step-compare process for the next time-step, allowing us to track entire trajectories. Architecting the grading system this way gives two major advantages: first, we avoid problems with drift due to the nature of floating point arithmetic, and second, we can pinpoint the exact instances when the implementations diverge by more than the acceptable tolerance, allowing us to provide detailed, instant visual feedback to guide students to the sources of their errors.

**Instant Feedback.** We provide different feedback to students depending on the information at hand and on the type of simulation under study. In all simulations, we examine the position and velocity of the system, highlighting objects in which the oracle and the student implementations disagree. Errors in the core state of a simulation typically imply that a student has incorrectly implemented an integrator or a force, thus giving them a concrete, actionable avenue for debugging their code. For systems that involve collisions between objects, simulations have additional potential failure modes. Collision detection – the process of both computing what objects overlap and of extracting useful information from this overlap – has two potential failure modes: a student can fail to detect a collision, or a student can compute incorrect information from the collision. With the numerical oracle, we provide visual indicators of both scenarios. By providing distinct visual indicators for each failure case, we provide more precise feedback to students to assist them in fixing their code.

### 4.2 Assignment Structure and Grading

**Autograded assignments.** With a numerical auto-grader in hand, a structure for course assignments and a grading policy follow naturally (Fig. 4.3). The course is divided into five major themes, each covering a general topic of importance in physically based animation: mass-spring systems, collision response, rigid body simulation, fluid simulation, and a final theme focusing on the creative aspects of computer graphics. These themes are further divided into weekly milestones, which focus on a specific aspect of the current theme. Course lectures
Figure 4.3: Autograder in practice: A student implements a course assignment, and runs an instructor-provided simulation description through their simulation code. The student can then load this simulation output and the simulation description into the autograder, which compares their simulation output against a known, vetted implementation of the algorithm under study. The autograder provides feedback to help the student debug the assignment if their implementation is incorrect. Finally, when the assignment is due, the student’s implementation is tested with a large corpus of input simulation descriptions using the autograder, and their final grade is determined by the number of simulation descriptions that are successfully performed by the student’s code.

introduce the material contained in each milestone, providing a high level overview of the topic at hand and overall perspective. We provide our full course notes in Chapter 5.

With each milestone, students add features to a unified simulation codebase, for which we furnish starter boilerplate code. The features required in each milestone are described in detail at the start of the assignment, similar to the exposition one would find in a textbook or in full course notes. This codebase reads input simulation description files in a documented format, which include information such as simulation initial conditions, forces, integrators, and rendering settings. The second section of each milestone describes how these features fit into the simulation description format and what options are valid for each of these features. The third section of each milestone describes the particular features that each student must implement,
and points the students to the portion of the codebase in which they should implement these features. Many features require additional mathematical derivations on the part of the student, allowing us to test students’ theoretical understanding of material as needed. For each feature, we provide half of the scenes with which we will test a student’s code, allowing the student to immediately engage with the assignment. We do not expose the remaining test scenes, encouraging students to thoroughly vet and test their implementations.

**Creative assignments.** At the conclusion of each milestone, we require the submission of a *creative scene*. This creative scene is submitted using the same simulation description format as the graded simulation scenes. For the creative scenes, we allow students to customize the simulation’s behavior through numerous provided callbacks that expose the internals of the overall simulation codebase. Students are encouraged to make these scenes as complicated as they desire and to also draw on whatever sources they would like for inspiration, although we emphasize that the creative submissions should showcase the features of the current milestone. Each creative assignment factors into the student’s score on the milestone in the same way as a simulation test scene, but exemplary submissions have the opportunity to earn roughly 15% extra credit per milestone. In addition, to celebrate excellent creative scenes, we demo the best scenes in front of the class, weekly at the conclusion of each milestone.

**Freeform assignments.** While the autograder based assignments have proven quite successful in increasing the pace of the course, we also like to ascertain how students perform when given less direction. To this end, we have a milestone where the goal is not to introduce new features to the simulation codebase, but to instead optimize a core feature of the codebase by designing and implementing a fast collision detection data structure. To help students with this element of the process, we provide both an introduction to broad phase collision detection data structures in the lectures and suggested readings, but students are otherwise free to implement whatever data structure they think will be most efficient in whatever manner they see fit. Each time a student submits a new version of their data structure, we run a series of benchmarks and update a live leader board with anonymized performance rankings. Each student’s grade for this milestone is based on relative performance to the best score in the class and to a base reference implementation of the data structure.

Expanding on the notion of a freeform theme, we include an entire freeform milestone in
which students implement a two dimensional fluid simulator from scratch. While we provide a boilerplate project with simple user interface code, students are required to read a significant publication [Stam, 1999] in the physically based animation literature, and implement a simulation based on their understanding of this paper. We do not grant students access to the autograder for this assignment. This final technical milestone serves as a test of a student’s independence, accumulated mathematical prowess, and the ability to independently write and debug numerical-intensive code.

**Late policy.** Owing to the cumulative nature of our course material – assignments typically build on the material in previous assignments – we release a reference implementation for the previous milestone at the start of each new milestone. As a side effect of this solution release and the weekly schedule of milestones, we must impose a hard cutoff date for late submissions of all milestones. Late assignments lose 1% of their achieved score for every six minutes of lateness, allowing us to release the next milestone 10 hours after the conclusion of the previous milestone. To account for any difficulty stemming from this policy, we drop each student’s lowest milestone grade in the final grade computation.

### 4.3 Implementation Details

All course assignments are implemented with the C++ programming language. While this course is many students first exposure to C++, we specifically selected the language due to its ubiquity within the computer graphics community. To ease the difficulties that students new to C++ may encounter, we provide extensive code for the user interface, boilerplate code for the overall framework’s structure, and a complete build system. The focus is thus on implementing the fundamental algorithms of physically based animation rather than software engineering in C++. The simulation scene format is implemented with XML, as the format is both human readable and easily parsed in most programming languages. We encourage students to use existing programming tools to build their creative scenes, and we also provide starter scripts in the Python programming language to help with common tasks such as mesh processing and XML input and output.

Students submit their assignments with a Python script that runs on the department’s
Linux infrastructure. We also provide a standard virtual machine with the necessary library dependencies pre-installed and with the full numerical autograder executable. Students are allowed to develop their software on their platform of choice, but their final submissions must run on the standard departmental Linux installation.

4.4 Future Work

We are excited about the potential use of technology to further smooth the autograding process. An assignment submission system based on the Git version control system would bring many benefits: students would be encouraged to version their assignments and to break their submissions into bite sized deliverables, students would gain experience with real world tools that they could apply elsewhere, and building the submission system on industry-standard tools would lead to a less brittle and more maintainable infrastructure from the instructor’s standpoint. We are also excited by the prospect of easing the implementation of students’ creative scenes by exposing customization callbacks in a higher level language like Python, allowing for both faster iteration and access to a huge number of libraries that students could use when constructing their scenes. We are interested in expanding the course curriculum to include a wider range of topics, including the simulation of volumetric elastica and automated control of simulations. Finally, inspired by the success of interactive, online course offerings from Khan Academy [Khan, 2018] and Pixar in a Box [Pixar, 2018], we are interested in a web-based (JavaScript and WebGL) variant of our autograding oracle and simulation codebase. While this web-based variant of the course would not reflect the industry standard tools used in physically based animation, it would significantly reduce the startup cost incurred by students of the course.
Chapter 5

COMS 4167: Physically Based Animation

In this chapter, we present the course notes we developed for the inaugural semester of COMS 4167 that employed our autograding oracle. We divided the course into four general technical themes: mass-spring systems, collisions, rigid body simulation, and fluid simulation. As detailed below, the mass-spring, collision, and rigid body themes are further divided into subthemes that focus on a subset of the topics in the area of study. At the conclusion of the course, we ask students to complete a final creative assignment in which they use the accumulated technical material from all previous themes to design and execute a substantial capstone simulation.

5.1 Mass-Spring Systems: Theme I Milestone I

5.1.1 Introduction

Welcome to COMS 4167, Physically Based Computer Animation! In this theme, you will implement a particle system that supports a number of forces and integrators. This theme is divided into three graded milestones due a week apart. Prior to each milestone, you will be given starter code from which you should build your milestone. This starter code will contain a complete implementation of the previous milestone. You will also have access to both a grading oracle that you can benchmark your program’s output against, as well as half the examples
that will be used to grade your program.

5.1.2 Academic Honesty Policy

You are permitted and encouraged to discuss your work with other students. You may work out equations in writing on paper or a whiteboard. You are encouraged to use the Wiki bulletin board to converse with other students, the TA, and the instructor.

HOWEVER, you may NOT share source code or hardcopies of source code. Refrain from activities or the sharing of materials that could cause your source code to APPEAR TO BE similar to another student’s enrolled in this or previous years. We will be monitoring source code for individuality. Cheating will be dealt with severely. Cheaters will be punished. Source code should be yours and yours only. Do not cheat.

5.1.3 Grading and Lateness

Each milestone will be tested with a fixed number of example problems. Each of these example problems is graded on a pass/fail basis, and your final grade for the milestone is the percentage of problems that passed. You will have access to half of the grading problems before the milestone is due, as well as the grading program itself. Each milestone will count equally towards your final grade.

Late submissions lose 1% per six minutes of lateness. For example: a submission that is two hours late is penalized 20%, and a submission that is ten hours late receives no credit. Rationale: Since weekly milestones build on each other, we must ensure that all students begin each week on an equal playing field. By enforcing a strict lateness policy, we will be able to post the solution to each milestone shortly after it is due, thereby enabling students to build on a solid foundation in the following week.

Plan ahead. The only exception to this policy is a documented medical emergency. In order to ensure fair grading, exceptions are not possible for holidays, sport meets, theater appearances, indigestion, etc. Plan ahead. If you believe that you have a just cause for submitting a late assignment in a non-medical-emergency circumstance, please obtain written permission from the instructor or TA at least one week prior to the assignment deadline. Plan ahead.
5.1.4 Obtaining and Building the Starter Code

Approximately one week before each milestone is due, new C++ starter code will be posted to the course wiki. This code has been designed and tested in a Unix environment (Linux, BSD, macOS, etc) with the GCC toolchain - you are welcome to develop code on the platform of your choice, but your final submission MUST compile and run on clic.cs.columbia.edu. To obtain and build the starter code:

1. Ensure that cmake is installed on your development system. On a Linux or BSD distribution, cmake can be installed through the distribution’s package manager (e.g. apt-get, rpm, or port). On a macOS installation, cmake is available through Homebrew package distribution system. Stand alone installers are also available for all major platforms from the cmake [download page](https://cmake.org)

2. Download Theme01Starter01.zip from the course wiki

3. Extract the archive, and rename the directory Theme01Starter01 to your UNI (e.g. bds2114)

4. From the root of this archive, create a build directory

5. Change into the build directory, and execute the command

   ```
cmake .
   ```

6. Configure the build system by executing the command below. In the resulting configure menu, you will have the option of setting the build mode. During development, you might find it useful to test your program with the build mode set to Debug. During grading, your program will be tested in Release mode, so please ensure your code runs in Release mode before the final submission.

   ```
   ccmake .
   ```

7. After setting options, press c to save the configuration, and g to update the build system and exit
8. Build the program by executing

```
make
```

9. The executable is now in the build/FOSSSim directory. Change into this directory, and execute the command

```
./FOSSSim -s assets/helloworld.xml
```

You can add additional source files to your program by placing them in the FOSSSim directory under the project root. You will have to regenerate the build system whenever you add new source files.

### 5.1.5 Test Scene Oracle

To aid you during development, you will have access to a precompiled oracle implementation of this theme. This oracle is the same program your submission will be graded with. As you will have access to half of the test examples, you will know your exact grade on half of the milestone before the due date. The other examples will remain hidden, but will be similar in nature to those you are provided. The additional examples may exploit combinations of forces or parameters not covered by the set you have access to. The intent of keeping these hidden is to encourage you to thoroughly test your code on your own example problems.

The oracle program can load the output of your simulation and visually highlight areas in which your simulation is incorrect. The oracle will also tell you if the particular simulation is judged a success. In addition to the example problems, the oracle will function with any scene of your design that adheres to the standard (see the XML File Format section).

You have two options for benchmarking your code: a ‘live’ OpenGL display, and ‘headless’ svg output.

#### 5.1.5.1 Benchmarking Simulations Against the Oracle with OpenGL Feedback

1. Log into one of the CS CLIC computers using your CS account. For example, from a bash terminal using x forwarding, execute:
ssh -X your_cs_login@clic.cs.columbia.edu

2. Run your program with binary output enabled. For example, execute:

   ./FOSSSim -s scene_file.xml -o binary_output.bin

3. Run the oracle program with the same scene file in input mode. For example, execute:

   /home/smith/4167/FOSSSimOracle -s scene_file.xml -i binary_output.bin

4. If the oracle detects any errors, it will circle the offending particles on the screen. After
   executing the scene, the oracle will print the total position residual, the total velocity
   residual, the maximum position residual, and the maximum velocity residual. This output
   will indicate whether the detected residuals are acceptable.

5.1.5.2 Benchmarking Simulations Against the Oracle with Headless SVG Feedback

1. Log into one of the CS CLIC computers using your CS account. For example, from a
   bash terminal, execute:

   ssh your_cs_login@clic.cs.columbia.edu

2. Build your program on the CLIC computer.

3. Run your program in headless mode with binary output enabled. For example, execute:

   ./FOSSSim -s scene_file.xml -o binary_output.bin -d 0

4. Create a directory in which you would like to generate snapshots of your simulation. Run
   the oracle program with the same scene file in input mode, headless mode, and ‘movie’
   mode. For example, execute:

   /home/smith/4167/FOSSSimOracle -s scene_file.xml -i binary_output.bin
   -d 0 -m output_dir
5. The directory output_dir will contain snapshots of each step of your simulation. If the oracle detected any errors, the offending particles will be circled in red. After executing the scene, the oracle will print the total position residual, the total velocity residual, the maximum position residual, and the maximum velocity residual. This output will indicate whether the detected residuals are acceptable.

6. If you would like to turn the collection of svg files into a movie, we have provided a python script, `generatemovie.py`, on the course wiki. Execute this script with no arguments to obtain instructions.

5.1.5.3 Running the Grading Program

To execute the grading program, place your copy of the milestone into a zip file. Execute the command

```python
python /home/smith/4167/gradeassignment.py assignmentdir scenedir oracleprogram gradedir
```

where assignmentdir is the directory containing your zipped up milestone and ONLY your zipped up milestone, scenedir is a directory containing the test scenes (you should use the assets directory from the starter code), oracleprogram is the path to the grading program listed above, and gradedir is the directory you would like to place the grade report in.

**Note that your submission MUST work with the grading program.** This is a hard requirement, and no exceptions will be made. Therefore, if you encounter any problems, please seek out help as soon as possible.

Furthermore, the most serious infraction that you can commit in this class is to try to outwit the grading program: do not write source code or script files that could be interpreted as trying to circumvent the intent behind the automatic grader—this will result in the most severe penalty to your academic standing.

Please note that your program will be graded using this same oracle and script.
5.1.6 Virtual Machine Development Environment

We have posted a Linux virtual machine capable of running the starter code to the “Development Environment” section of the course wiki. This virtual machine runs with the free virtualization software Virtual Box, which you can download at http://www.virtualbox.org/. We have tested this virtual machine in Ubuntu Linux and the latest releases of Windows and macOS. The login for this virtual machine is comsw4167 and the password is 1123581321. You will still require access to clic.cs.columbia.edu in order to test your code with the grading oracle. Your code MUST work on the clic computers with the provided grading framework, no exceptions. Therefore, please ensure that you can access these computers with your cs login.

5.1.7 XML File Format

All scenes in this program are specified using an xml file. The features supported in the first milestone are defined below. XML parsing code has been provided for you.

1. The root node of the file is the scene node:

   <scene>
     ... scene contents ...
   </scene>

2. The duration of the simulation is specified with the duration node:

   <duration time="10.0"/>

   The time attribute is a scalar that specifies how long the scene should execute in ‘simulation seconds.’

3. The integrator attribute specifies both the integrator to solve the system with as well as a time-step:

   <integrator type="explicit-euler" dt="0.01"/>

   For the first milestone, the integrator type will always be explicit-euler. The scalar attribute dt specifies the time-step to use with the given integrator.
4. The particle node adds a particle to the system:

```xml
<particle m="1.0" px="1.0" py="7.5" vx="0.2" vy="-0.3" fixed="0"
    radius="0.04"/>
```

The scalar m attribute specifies the mass of the particle. The scalar px and py attributes specify the initial position of the particle. The scalar vx and vy attributes specify the initial velocity of the particle. The boolean fixed attribute specifies whether this particle is fixed (not-simulated) or free (simulated). The optional scalar radius attribute specifies the particle’s radius; if radius is not specified, a default value is used.

5. The edge node creates an edge between two particles:

```xml
<edge i="1" j="2" radius="0.01"/>
```

The integer i and j attributes specify the particles that compose the edge. The optional scalar radius attribute specifies a radius for the edge.

6. The simplegravity node defines a constant gravitational force:

```xml
<simplegravity fx="0.0" fy="-9.81"/>
```

The scalar fx and fy attributes define the x and y components of gravity, respectively.

7. The maxsimfreq node defines a maximum frequency at which to step the system in interactive mode. This allows you to see simulations that would otherwise run too quickly.

```xml
<maxsimfreq max="500.0"/>
```

The scalar attribute max defines the maximum simulation frequency.

8. The particlecolor node changes a particle’s color.

```xml
<particlecolor i="2" r="0.1" g="0.2" b="0.3"/>
```

The integer attribute i identifies the particle. The scalar r, g, and b attributes set the particle’s color. The r, g, and b attributes must have values between 0.0 and 1.0.
9. The edgecolor node changes an edge’s color.

\[
\text{<edgecolor i="4" r="0.2" g="0.3" b="0.4"/>}
\]

The integer attribute i identifies the edge. The scalar r, g, and b attributes set the edge’s color. The r, g, and b attributes must have values between 0.0 and 1.0.

10. The particlepath node causes a particle to trace out a colored path during simulation.

\[
\text{<particlepath i="6" duration="10.0" r="1.0" g="0.9" b="0.8"/>}
\]

The integer attribute i identifies the particle. The duration attribute specifies how many simulation seconds each point on the path lasts. The scalar r, g, and b attributes set the path’s color. The r, g, and b attributes must have values between 0.0 and 1.0.

Future milestones and themes will define additional features.

5.1.8 Required Features for Milestone I

5.1.8.1 Explicit Euler

Recall from class that one can discretize Newton’s second law using explicit Euler, giving:

\[
\begin{align*}
q^{n+1} &= q^n + h\dot{q}^n \\
\dot{q}^{n+1} &= \dot{q}^n + hM^{-1}F(q^n, \dot{q}^n)
\end{align*}
\]

Observe that both the position and the velocity update depend only on the position and velocity at the previous time-step. Edit the provided source file `ExplicitEuler.cpp` to compute the updated position and velocity using explicit Euler.

The kinetic energy of a particle is given by \( T = \frac{1}{2}mv^2 \). Here a boldface font denotes a vector quantity. Recall that the dot product between two vectors \( \mathbf{a} \) and \( \mathbf{b} \) can be computed as \( \mathbf{a} \cdot \mathbf{b} = a_x b_x + a_y b_y \) (here \( a_x \) denotes the x component of the vector \( \mathbf{a} \)). Furthermore, the shorthand \( v^2 = v \cdot v \) is employed. Edit the provided source file `TwoDScene.cpp` to compute the kinetic energy of the system. Newton’s first law implies that in the absence of external forces, the kinetic energy will be constant. As a simple ‘sanity’ check of your integrator, execute the
scene InertiaTests/test01explicit.xml and print the kinetic energy at each time-step to verify that it is indeed constant and the expected value. Write code to save the kinetic energy to a text file with the format (there are ‘stubs’ in TwoDScene.cpp and main.cpp):

```
# Time KineticEnergy
time0 energy0
time1 energy1
... ...
```

A python script that will generate a plot form this file has been posted to the course wiki; executing the script will print instructions for its use. Please verify that the kinetic energy is constant. These plots will not be graded, but you should get in the habit of debugging your programs both textually and graphically – it will pay off.

### 5.1.8.2 Constant Gravity

Recall from introductory physics that, sufficiently close to earth’s surface, we can approximate gravity’s effect as a constant acceleration \( g \) on all objects. Placing the 0 potential reference at the origin, this force corresponds to a potential energy of \( U(x) = -mg \cdot x \). Taking the gradient of this potential, the force is given by \( F = -\nabla U = mg \).

Edit the provided source file SimpleGravityForce.cpp to compute this potential energy and its gradient.

### 5.1.8.3 Fixed Degrees of Freedom

During a simulation, it is often useful to fix or kinematically script a degree of freedom. Later in the course we will discuss methods for enforcing constraints in your simulations, but for now a simple solution is to just set the force for that degree of freedom to 0. Add this functionality to your explicit Euler implementation.

### 5.1.8.4 Creative Scene

As part of your final submission for this milestone, please include a scene of your design that best shows off your program. This scene will count towards the final grade the same as the standard
test scenes. Based on the quality of your scene, however, you will have the opportunity to earn up to 15% extra credit. Your scene will be judged by a secret committee of top scientists using the highly refined criteria of:

1. How well the scene shows off this milestone’s ‘magic ingredients’ (a la Iron Chef).
2. Aesthetic considerations. The more beautiful, the better.
3. Originality.

Top examples will be posted to the course wiki, and possibly demoed for the class. To aid you in constructing more complex scenes, a collection of python scripts have been posted to the wiki.

To submit this scene, simply place it in the CreativeScenes directory of your submission. Please name your scene file youruniTheme01Milestone01.xml where youruni is your uni.

5.2 Mass-Spring Systems: Theme I Milestone II

5.2.1 Introduction

In Milestone II of Theme I, you will implement a new integrator, forward-backward Euler, as well as spring, gravitational, and damping forces. Roughly five days before the due date, we will post a reference implementation of Milestone I. The grading and lateness policies remain the same as in Milestone I; you will have access to an ‘oracle’ and roughly half of the testing scenes.

5.2.2 Clarification on Fixed Degrees of Freedom

Clarifying the language used in Milestone I, a vertex (used synonymously with particle) has two degrees of freedom, an x coordinate and a y coordinate. When we refer to ‘fixing’ a particle, we mean that this particle should remain in the location given by its initial position in the XML file. To achieve this, one can set the initial velocity of the particle to 0 (we guarantee that all fixed particles in all provided test scenes will start with an initial velocity of 0) and ensure that no forces act on the particle. One way to achieve this is by setting the entries corresponding to the fixed particle’s degrees of freedom to 0 in the force vector.
For example, if particle 3 were fixed, after computing all forces, one would set entries 6 and 7 in the global force vector to 0.

### 5.2.3 New XML Features

In addition to the xml tags from Milestone I, Milestone II adds the new features:

1. The *integrator* node now accepts the type “forward-backward”:

   ```xml
   <integrator type="forward-backward" dt="0.01"/>
   ``

2. The *springforce* node adds a spring to the system:

   ```xml
   <springforce edge="0" k="2.0" l0="1.5" b="0.1"/>
   ``

   The edge property sets the edge this spring is associated with, the k attribute sets the stiffness of the spring, and the l0 attribute sets the rest length of the spring. The optional property b introduces an internal damping force to the spring. If b is not specified, it defaults to 0.

3. The *gravitationalforce* node adds an attractive force acting between two particles:

   ```xml
   <gravitationalforce i="0" j="1" G="0.000118419"/>
   ``

   The properties i and j set the particles the force acts on, and the G attribute linearly scales the magnitude of the gravitational force.

4. The *dragdamping* node adds a force to the system that resists the motion of all particles:

   ```xml
   <dragdamping b="3.0"/>
   ``

   The property b is a constant that linearly scales the magnitude of the damping force.

5. The *scenetag* node adds a string name to the scene that can be accessed from main:

   ```xml
   <scenetag tag="RiemannRocks"/>
   ```
CHAPTER 5. COMS 4167: PHYSICALLY BASED ANIMATION

Figure 5.1: Geometry of particle interactions: The separation distance and the normalized separation vector used to compute pairwise spring and gravitational forces between particles.

This tag is available from main as the $g\_scene\_tag$ variable.

6. The $tag$ attribute of the $particle$ node associates a string tag with a particle:

```xml
<particle m="1" px="0" py="0.6" vx="0" vy="0" fixed="0" tag="string"/>
```

These tags are stored in the $m\_particle\_tags$ vector in TwoDScene. A reference to this vector can be obtained with the $getParticleTags()$ method.

5.2.4 Required Features for Milestone II

5.2.4.1 Spring Force

The potential energy of a spring, or ‘harmonic oscillator,’ is given by:

$$U(x_i, x_j) = \frac{1}{2} k (l(x_i, x_j) - l_0)^2$$

Let $l(x_i, x_j) = \sqrt{(x_i - x_j)^2}$ denote the distance between the two particles, let $l_0$ denote the spring’s rest length, and let $k$ denote the spring’s stiffness. Note that $l_0$ is typically a constant, although one could ‘script’ this parameter’s value to achieve an artistic effect (for example, one could increase $l_0$ to achieve an inflation-like effect). See Figure 5.1. Computing the gradient of the potential with respect to one of the particles, we find by the chain rule that

$$\nabla x_k U = k (l - l_0) \nabla x_k l.$$
Computing $\nabla_{x_k} l$, we find that

$$\nabla_{x_k} l = \frac{1}{2} \left((x_i - x_j)^2\right)^{-\frac{1}{2}} 2(x_i - x_j) \nabla_{x_k} (x_i - x_j) = \nabla_{x_k} (x_i - x_j) \frac{x_i - x_j}{\sqrt{(x_i - x_j)^2}} = \nabla_{x_k} (x_i - x_j) \hat{n}$$

where $\hat{n} = (x_i - x_j)/\sqrt{(x_i - x_j)^2}$ is a unit-length vector pointing from vertex $x_j$ to vertex $x_i$.

For vertex $i$, $\nabla_{x_i} (x_i - x_j) = 1$, and for vertex $j$, $\nabla_{x_j} (x_i - x_j) = -1$.

This tells us that the gradient of the distance between two particles is parallel to a vector between the two particles (e.g. $\hat{n}$). Let us take a step back and see what we can learn from this result. Consider the interpretation of the gradient as the direction of maximum change. If we move a particle in either direction perpendicular to $\hat{n}$, the length between the particles will increase. This implies that the particle is at a local minimum along the direction perpendicular to $\hat{n}$, and thus that the directional derivative in this direction is 0. Therefore, the gradient, or direction of maximum change, must point along $\hat{n}$, and our result makes intuitive sense.

We conclude that the gradients of the potential with respect to each vertex are given by:

$$\nabla_{x_i} U = k(l - l_0) \hat{n}$$

$$\nabla_{x_j} U = -k(l - l_0) \hat{n}$$

Observe that the force (minus the gradient, don’t forget that pesky minus sign!) is directed along the vector between the particles, as we would expect for a spring. Further, observe that the gradients, and thus forces, sum to 0. That is, our solution obeys Newton’s third law.

Edit the provided source file $SpringForce.cpp$ to compute this potential energy and its gradient. Please make sure you add the force’s contribution to the proper location in the ‘global’ force vector. For example, if the spring acts on particles 2 and 5, you will add contributions to the 4th, 5th, 10th, and 11th entries of the ‘global’ force vector.

### 5.2.4.2 Gravitational Force

The potential energy of two particles interacting via gravity is given by:

$$U(x_i, x_j) = -\frac{Gm_imj}{l}$$

As before, let $l(x_i, x_j) = \sqrt{(x_i - x_j)^2}$ denote the distance between the two particles. Computing the gradient of the potential with respect to one of the particles, we find by the chain rule that

$$\nabla_{x_k} U = \frac{Gm_imj}{l^2} \nabla_{x_k} l$$
where $\nabla_{x_k}l$ is again given by $\nabla_{x_i}l = \hat{n}$ and $\nabla_{x_j}l = -\hat{n}$. Thus, the gradients of the potential with respect to each vertex are given by:

$$\nabla_{x_i}U = \frac{Gm_i m_j}{l^2} \hat{n}$$

$$\nabla_{x_j}U = -\frac{Gm_i m_j}{l^2} \hat{n}$$

As a ‘sanity check’ we immediately observe that this force is central and obeys Newton’s third law (Newton’s third law states the familiar ‘for every action there is an equal and opposite reaction’).

Edit the provided source file *GravitationalForce.cpp* to compute this potential energy and its gradient.

### 5.2.4.3 Linear Damping Force

All of the forces we have introduced thus far are conservative; that is, we can assign a scalar valued potential to each point in space, and the difference in potential between any two points is independent of the path taken between the points. There are useful forces that are not conservative, however, such as friction and drag. Note that a damping force, by its very (dissipative) nature, is not conservative, i.e., it does not act to preserve total energy. You will implement one such force that linearly resists a particle’s motion. That is, for each particle in your system, the force’s magnitude is given by

$$F_i = -\beta \nu_i$$

where $\beta$ is a scalar damping constant.

Edit the provided source file *DragDampingForce.cpp* to compute this force. Watch out for sign errors!

### 5.2.4.4 Spring Damping Force

The linear damping force models the motion of an object in a ‘thick’ fluid, and will eventually damp ALL motion in the scene. In contrast, we can introduce a force that models internal dissipation within a spring. Unlike the linear damping force, this internal force only damps
motion that compresses or extends the spring. Given two particles $i$ and $j$ interacting with a spring force, the spring damping force is given by

$$F_i = -\beta \hat{n} \cdot (v_i - v_j) \hat{n}$$

$$F_j = \beta \hat{n} \cdot (v_i - v_j) \hat{n}$$

where $\beta$ is a constant to scale the magnitude of the damping force, and $\hat{n}$ is defined as in the $SpringForce$ section. Note that while the spring damping force is not conservative (and it leads to a loss of energy), it does obey Newton’s third law, and therefore it does conserve the total momentum of both particles.

Augment the spring force in the source file $SpringForce.cpp$ to include this damping force.

### 5.2.4.5 Forward-Backward Euler

As an alternative to explicit Euler we can discretize Newton’s second law using forward-backward Euler. This yields the ‘update rule’ of:

$$\dot{q}_{n+1} = \dot{q}_n + hM^{-1}F(q^n, \dot{q}^n)$$

$$q_{n+1} = q^n + h\dot{q}_{n+1}$$

Notice that the velocity update depends only on the position and velocity at the previous
time-step, while the position update depends on the velocity at the current time-step. Contrast this with explicit Euler, where both updates depend on the previous step’s position and velocity.

To qualitatively verify the behavior of forward-backward Euler, compare to the behavior of explicit Euler with the test examples assets/GravityTests/test00explicit.xml and assets/GravityTests/test00forwardbackward.xml. Explicit Euler should produce an unstable orbit and ‘spiral outward,’ while forward-backward Euler should produce a stable orbit. For your personal edification, plot and compare the total energy with explicit Euler and forward-backward Euler. See Figure 5.2 for examples of these orbits.

As an additional test, run the scene assets/GravityTests/test00explicit.xml. For the default time-step and spring stiffness, you should simply see an oscillating spring. Experiment with different spring stiffnesses, masses, and time-steps. For large values of $\frac{k}{m}$ and for large time-steps, you should observe ‘explosions’ with explicit Euler. Conduct these same tests with forward-backward Euler.

Implement forward-backward Euler in the provided file ForwardBackwardEuler.cpp.

5.2.5 Scene Scripting

We have added the ability to assign tags to particles and scenes from an XML file. See the XML features section for a description of these features. You can use the scene tag from main to identify the scene, and the particle tags from TwoDScene to identify individual particles. We have also added a callback to main named sceneScriptingCallback() that is called after each time-step. Using this function and the tags, you can add custom events to your creative scene. We have provided code that simulates a water fountain by teleporting particles using these features; with the starter code, execute ParticleFountain.xml to see this example in action. Note that none of the graded test scenes will use these features, they just exist to help you develop exciting creative scenes.
5.3 Mass-Spring Systems: Theme I Milestone III

5.3.1 Introduction

In Milestone III of Theme I, you will implement a new integrator, linearized implicit Euler. This will involve the computation of the force Jacobian for each of the forces you implemented in Milestone II, as well as the solution of a linear system. Roughly five days before the due date, we will post a reference implementation of Milestone II. The grading and lateness policies remain the same as in previous milestones; you will have access to an ‘oracle’ and roughly half of the testing scenes.

You will also have the opportunity to earn extra credit by implementing a fully nonlinear version of implicit Euler, as well as by experimenting with a new vortex-generating, or ‘Biot-Savart,’ type force.

5.3.2 New XML Features

In addition to the xml tags from Milestones I and II, Milestone III introduces a new required feature:

- The integrator node now accepts the type “linearized-implicit” which is used in the feature linearized implicit Euler discussed below:

  \[
  \text{<integrator type="linearized-implicit" dt="0.01"/>}
  \]

Milestone III introduces a new extra credit feature:

- The integrator node now accepts the type “implicit” which is used in the extra-credit feature implicit Euler discussed below:

  \[
  \text{<integrator type="implicit" dt="0.01"/>}
  \]

We have also provided a ‘free’ feature that you are welcome to use while constructing your creative scene:
Figure 5.3: Newton’s method: The geometry of Newton’s method illustrated in one dimension. Linearizing the function around \( x_i \) and intersecting the line with the horizontal axis yields the next iterate \( x_{i+1} \).

- The vortexforce node defines a ‘vortex force’ acting between two particles, which we have implemented and is used in the ‘free’ feature we discuss in the vortex force section:

\[
\text{<vortexforce i="0" j="1" kbs="0.8" kvc="1000.0"/>}
\]

\( i \) and \( j \) specify the particles this force acts between. \( kbs \) controls the strength of the overall ‘Biot-Savart’ force, while \( kvc \) controls the amount of viscous drag. You will find that this force works best when one particle’s mass is extremely large while the other is very small. The extremely massive particle will generate a vector field through which the lighter particle advects.

### 5.3.3 Introduction to Newton’s Method

Before discussing implicit Euler, we will briefly review root finding, Newton’s method for univariate problems, and Newton’s method for multivariate problems.

#### 5.3.3.1 Univariate Root Finding

Recall that the root of an equation \( f(x) \) is a value of \( x \) for which \( f(x) = 0 \). As a simple example, consider the linear function \( f(x) = 3x + 6 \). \( f(x) \) has a single root, \( x = -2 \). For an example with
multiple roots, consider the polynomial \( f(x) = x^2 - x - 6 \). You might recall from introductory algebra that we can solve for the roots of this equation in a number of ways: we could employ the quadratic formula, we could factor the polynomial, etc. Taking the later approach, we find that \( f(x) = x^2 - x - 6 = (x - 3)(x + 2) \), from which we immediately read off the roots \( x = 3 \) and \( x = -2 \). Graphically, these roots correspond to points where the parabola intersects the \( x \) axis. While similar methods exist for cubic and quartic polynomials, they do not exist in general for higher degree polynomials. Thus, one is forced to develop specialized algorithms to locate these roots.

Polynomial root finding is an important but specialized subclass of the more general nonlinear root finding problem. For example, we might want to find the root(s) of the nonlinear equation \( f(x) = e^x - x - 1 \). To do so, we will employ Newton’s method, which is really just an application of the venerable Taylor’s Theorem. By Taylor’s Theorem, we know that we can approximate a sufficiently smooth function as a series of polynomials. If we truncate this series, Taylor’s Theorem tells us that we make an error that increases with the distance from the point about which we compute the Taylor series. Practically speaking, this means we can approximate a function locally by a polynomial provided we don’t stray too far from the point of interest.

Newton’s method exploits Taylor’s Theorem by assuming we have some educated estimate of a root. If this estimate is sufficiently close to the true root, then we are not making a large error by approximating the nonlinear function as a line. Therefore, the root of this line, while not exactly equal to the root of the nonlinear function, will be an improvement on our current guess. We can repeat this process with the improved estimate, and compute an even better approximation of the true root. Repeating this process, we have turned the problem of nonlinear root finding into a sequence of linear root finding problems. Let’s make this concrete with some math. Computing the Taylor expansion up to the linear term for some nonlinear function \( f(x) \) about our initial guess \( x_0 \), we find that

\[
    f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2} f''(\xi)(x - x_0)^2
\]

where \( \xi \in (x_0, x) \). Assuming \( x_0 \) is close to the true root, we can safely neglect the higher order
error term and equate with 0, giving
\[ f(x) \approx f(x_0) + f'(x_0)(x - x_0) = 0. \]
Solving for the root, we find that
\[ x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}. \]
Repeating this process until we are satisfied with our estimate, we obtain the iterative method
\[ x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}. \]
See Figure 5.3 for an illustration of one step of Newton’s method.

5.3.3.2 Multivariate Root Finding

We would now like to find a root of the function \( F(x) \) where \( F : \mathbb{R}^N \rightarrow \mathbb{R}^N \) and \( x \in \mathbb{R}^N \). Let us follow the same prescription as in the univariate case; we will derive a linear approximation to \( F \) that does not give the exact solution, but that is ‘easy’ to solve and will hopefully give an improved estimate of the solution. Computing the Taylor series of \( F(x) \) about some estimate of the solution \( x_0 \), we find that
\[ F(x) = F(x_0) + \nabla F(x_0)(x - x_0) + H.O.T. \]
where \( \nabla F(x) \) is the gradient of the function \( F \). As \( F(x) \in \mathbb{R}^N \), \( \nabla F(x) \in \mathbb{R}^{N \times N} \). The \( i,j \) entry of the gradient is given by \( \frac{\partial F_i}{\partial x_j} \). If we neglect the higher order terms (H.O.T.) and equate the function with 0, we find that
\[ F(x) \approx F(x_0) + \nabla F(x_0)(x - x_0) = 0. \]
We can solve this system to obtain an improved estimate of the root. Turning this into an iterative process:
\[ x_{i+1} = x_i - \nabla F(x_i)^{-1}F(x_i) \]
In both the univariate and multivariate cases, the convergence of Newton’s method depends on the quality of the initial estimate of the root. If the initial estimate is far from the root, there is no reason to expect a linear approximation to land us near a root, and Newton’s method is unlikely to converge.
Figure 5.4: Notation for implicit integration: A summary of the notation we use to derive implicit Euler.

5.3.4 Introduction to Implicit Euler

So far our discussion has been fairly abstract, and you might wonder how root finding relates to the topic of our course, physically based computer animation. Consider the following discretization of Newton’s Second law:

\begin{align*}
\mathbf{q}^{n+1} &= \mathbf{q}^n + h\mathbf{\dot{q}}^{n+1} \\
\mathbf{\dot{q}}^{n+1} &= \mathbf{\dot{q}}^n + h\mathbf{M}^{-1}\mathbf{F}(\mathbf{q}^{n+1}, \mathbf{\dot{q}}^{n+1})
\end{align*}

Let \( N \) denote the number of vertices in the system. Here \( \mathbf{M} \) is a \( 2N \times 2N \) mass matrix, in our case a diagonal matrix with masses on the diagonal (entries 0, 0 and 1, 1 are the mass of the first particle, entries 2, 2 and 3, 3 are the mass of the second particle, etc). Since the mass matrix is a diagonal matrix, we can store it as a vector. In our implementation, this vector is \( \mathbf{m} = [m_0, m_0, m_1, m_1, \ldots, m_N, m_N] \). The \( i^{th} \) entry of the mass matrix is simply \( m[i] \).

Notice that the new position depends on the new velocity, and the new velocity depends on the new position - we are unable to solve one without the other! If this system were linear, we would only have to solve a linear system. \( \mathbf{F}(\mathbf{q}^{n+1}) \) could be nonlinear, however. How do we solve a nonlinear system of equations? By Newton’s method!

Observe that we have \( 4N \) unknowns in our system, where \( N \) is the number of vertices. \( 2N \) of these unknowns are positions, and \( 2N \) are the corresponding velocities. Concatenate these
unknowns into one big vector, call it $y^{n+1}$:

$$y^{n+1} = \begin{pmatrix} q^{n+1} \\ \dot{q}^{n+1} \end{pmatrix}$$

Similarly, combine the above nonlinear equations into a single vector of length $4N$:

$$G(y^{n+1}) = G(q^{n+1}, \dot{q}^{n+1}) = \begin{pmatrix} P(q^{n+1}, \dot{q}^{n+1}) \\ Q(q^{n+1}, \dot{q}^{n+1}) \end{pmatrix} = \begin{pmatrix} q^{n+1} - q^n - h\dot{q}^{n+1} \\ \dot{q}^{n+1} - \dot{q}^n - hM^{-1}F(q^{n+1}, q^{n+1}) \end{pmatrix}$$

Computing the gradient for Newton’s method, as discussed in class, we obtain a $4N \times 4N$ matrix:

$$\nabla G = \begin{pmatrix} \frac{\partial P}{\partial q^{n+1}} & \frac{\partial P}{\partial \dot{q}^{n+1}} \\ \frac{\partial Q}{\partial q^{n+1}} & \frac{\partial Q}{\partial \dot{q}^{n+1}} \end{pmatrix} = \begin{pmatrix} \text{Id} & -h\text{Id} \\ -hM^{-1} \frac{\partial F}{\partial q^{n+1}} & \text{Id} - hM^{-1} \frac{\partial F}{\partial \dot{q}^{n+1}} \end{pmatrix}$$

A single step of Newton’s method now involves solving the linear system:

$$\nabla G(y^{n+1}_i)(y^{n+1}_i - y^{n+1}_i) = -G(y^{n+1}_i)$$

This system is repeatedly solved until convergence is detected. Observe that we attach a subscript to each occurrence of $q^{n+1}$ and $\dot{q}^{n+1}$ to denote that we are computing a sequence of these values. In contrast, the solution from the last time-step, $q^n$ and $\dot{q}^n$, is constant throughout this process and does not receive a subscript.

### 5.3.4.1 Reduction of System Size by Substitution

The above formulation is what we discussed in class. We could certainly proceed as described above, but we can reduce the size of our linear system with a bit of algebra (although this will modify the sparsity structure of the matrix). Let us expand a step of Newton’s method. Multiplying out the blocks $\nabla G(y^{n+1}_i)(y^{n+1}_i - y^{n+1}_i) = -G(y^{n+1}_i)$, we obtain two linear equations of size $2N$:

$$(q^{n+1}_i - q^n_i) - h(q^{n+1}_i - \dot{q}^{n+1}_i) = -(q^{n+1}_i - q^n_i - h\dot{q}^{n+1}_i)$$

$$-hM^{-1} \frac{\partial F(q^{n+1}_i, \dot{q}^{n+1}_i)}{\partial q^{n+1}_i}(q^{n+1}_i - q^n_i) + (\text{Id} - hM^{-1} \frac{\partial F(q^{n+1}_i, \dot{q}^{n+1}_i)}{\partial \dot{q}^{n+1}_i})(\dot{q}^{n+1}_i - \dot{q}^{n+1}_i) = -(\dot{q}^{n+1}_i - q^n_i - hM^{-1}F(q^{n+1}_i, q^{n+1}_i))$$
The first equation simplifies to
\[ q_{i+1}^{n+1} = q^n + h \dot{q}_{i+1}^{n+1} \]
which we can use to eliminate \( q_{i+1}^{n+1} \) from the second equation. Carrying out this algebra, we find
\[
\left( M - \left( h^2 \frac{\partial F}{\partial q^{n+1}} + h \frac{\partial F}{\partial \dot{q}^{n+1}} \right) \right) (\dot{q}^{n+1}_i - \dot{q}^{n+1}) = -M(\dot{q}^{n+1}_i - \dot{q}^n_i) + hF(q^n + h\dot{q}^{n+1}_i, \dot{q}^{n+1}_i)
\]
or relabeling \( \delta \dot{q}_{i+1} = \dot{q}_{i+1}^{n+1} - \dot{q}_i^n \)
\[
\left( M - \left( h^2 \frac{\partial F}{\partial q^{n+1}} + h \frac{\partial F}{\partial \dot{q}^{n+1}} \right) \right) \delta \dot{q}_{i+1} = -M(\dot{q}^{n+1}_i - \dot{q}^n_i) + hF(q^n + h\dot{q}^{n+1}_i, \dot{q}^{n+1}_i)
\]
(5.1)
where \( \frac{\partial F}{\partial q^{n+1}} \) and \( \frac{\partial F}{\partial \dot{q}^{n+1}} \) are evaluated at \((q^n + h\dot{q}^{n+1}_i, \dot{q}^{n+1}_i)\). Note that this \( 2N \times 2N \) system is half the size of our original \( 4N \times 4N \) system. After solving this linear system for \( \delta \dot{q}_{i+1} \) and computing \( \dot{q}_{i+1}^{n+1} = \dot{q}_i^n + \delta \dot{q}_{i+1} \), we compute \( q_{i+1}^{n+1} \) by the simple rule:
\[
q_{i+1}^{n+1} = q^n + h\dot{q}_{i+1}^{n+1}
\]

5.3.5 Linearized Implicit Euler

A common ‘optimization’ to implicit Euler employed in the graphics community is to not run Newton’s method to convergence on the fully nonlinear problem, but to instead linearize about the previous time-step’s solution and perform a single linear solve. This is equivalent to performing one iteration of Newton’s method with the initial iterate set to the previous time-step’s solution, that is with \( q_{0}^{n+1} = \dot{q}^n \). This results in the system
\[
\left( M - \left( h^2 \frac{\partial F}{\partial q^{n+1}} + h \frac{\partial F}{\partial \dot{q}^{n+1}} \right) \right) \delta \dot{q} = hF(q^n + h\dot{q}^n, \dot{q}^n)
\]
(5.2)
where \( \delta \dot{q} = \dot{q}^{n+1} - \dot{q}^n \) and where \( \frac{\partial F}{\partial q} \) and \( \frac{\partial F}{\partial \dot{q}} \) are evaluated at:
\[
\frac{\partial F}{\partial q} = \frac{\partial F(q^n + h\dot{q}^n, \dot{q}^n)}{\partial q}
\]
\[
\frac{\partial F}{\partial \dot{q}} = \frac{\partial F(q^n + h\dot{q}^n, \dot{q}^n)}{\partial \dot{q}}
\]
After solving for \( \delta \dot{q} \), we know \( \dot{q}_{i+1}^{n+1} = \dot{q}^n + \delta \dot{q} \). Once \( \dot{q}_{i+1}^{n+1} \) is known, \( q_{i+1}^{n+1} \) can be computed by:
\[
q_{i+1}^{n+1} = q^n + h\dot{q}_{i+1}^{n+1}
\]
5.3.6 Note on ‘Local’ and ‘Global’ Indices

In many of the computations below, you are provided formulae for force Jacobians in ‘local’ coordinates. For example, the spring force involves two particles or 4 degrees of freedom, and the corresponding force Jacobian is a $4 \times 4$ matrix. You will need to place these components into the ‘global’ force Jacobian, however.

Consider a system with three vertices, or 6 degrees of freedom. The force Jacobian will be a $6 \times 6$ matrix, but we can view it as a $3 \times 3$ ($\text{vertices} \times \text{vertices}$) matrix with $2 \times 2$ ($x, y \times x, y$) blocks. At any given instant, let the force Jacobian be given by

$$
\begin{pmatrix}
A & B & C \\
D & E & F \\
G & H & I \\
\end{pmatrix}
$$

where each letter denotes a $2 \times 2$ matrix. Consider a spring force connecting particles 0 and 2. This spring force will produce a force Jacobian given in ‘local’ indices by:

$$
\begin{pmatrix}
P & Q \\
R & S \\
\end{pmatrix}
$$

When we add this local force Jacobian into the global force Jacobian, we will obtain the matrix

$$
\begin{pmatrix}
A + P & B & C + Q \\
D & E & F \\
G + R & H & I + S \\
\end{pmatrix}
$$

5.3.7 Required Features for Milestone III

5.3.7.1 Linearized Implicit Euler

You will implement linearized implicit Euler as detailed in the preceding Linearized Implicit Euler section. This will involve computing the force Jacobians for the forces from Milestone II; we have provided these formulae below. In addition, you will have to solve a linear system. We have provided both code to solve a linear system as well as an example with a ‘toy’ linear system. Copying this example here:

```c
// Create a 10 x 10 matrix
```
MatrixXs A(10,10);
// Fill the matrix with random numbers
A.setRandom();
// Create a vector of length 10
VectorXs b(10);
// Fill the vector with random numbers
b.setRandom();
// Compute the solution to A*x = b
VectorXs x = A.fullPivLu().solve(b);
// Verify that we computed the solution (the residual should be roughly 0)
std::cout << (A*x-b).norm() << std::endl;

Please implement linearized implicit Euler in the provided source file LinearizedImplicitEuler.cpp; the linear solver example is contained in this file.

5.3.7.2 Simple Gravity Force Jacobian

The simple gravity force has a constant force Jacobian, that is
\[
\frac{\partial \mathbf{F}}{\partial \mathbf{q}} = \frac{\partial \mathbf{F}}{\partial \mathbf{\dot{q}}} = 0
\]

5.3.7.3 Spring Force Jacobian

For two particles in 2D interacting with a spring force, the force Jacobian is a symmetric 4 \times 4 matrix
\[
\frac{\partial \mathbf{F}}{\partial \mathbf{q}} = \begin{pmatrix}
K & -K \\
-K & K
\end{pmatrix}
\]
where \( K \) is a symmetric 2 \times 2 matrix given by
\[
K = -k \left( \hat{n} \hat{n}^T + \frac{l - l_0}{l} (\mathbf{I} - \hat{n} \hat{n}^T) \right).
\]
k denotes the spring stiffness, \( \hat{n} = (\mathbf{x}_i - \mathbf{x}_j)/|\mathbf{x}_i - \mathbf{x}_j| \) denotes a normalized vector pointing from \( \mathbf{x}_j \) to \( \mathbf{x}_i \), \( l = |\mathbf{x}_i - \mathbf{x}_j| \) denotes the length of the spring, \( l_0 \) denotes the rest length of the spring, and \( \mathbf{I} \) denotes the identity matrix. Recall that \( \hat{n} \hat{n}^T \) is an example of an outer product. Our spring force has no velocity dependence, so \( \frac{\partial \mathbf{F}}{\partial \mathbf{\dot{q}}} = 0 \).
Augment SpringForce.cpp to compute the force Jacobian.

5.3.7.4 Spring Damping Force Jacobian

The spring damping force depends on both its particles’ positions and its particles’ velocities. Therefore, both $\frac{\partial F}{\partial q}$ and $\frac{\partial F}{\partial \dot{q}}$ are nonzero. $\frac{\partial F}{\partial q}$ is given by:

$$\frac{\partial F}{\partial q} = \begin{pmatrix} K & -K \\ -K & K \end{pmatrix}$$

where $K$ is a $2 \times 2$ matrix given by:

$$K = -\frac{\beta}{l} (\hat{n} \cdot (v_i - v_j)I_d + \hat{n}(v_i - v_j)^T)(I_d - \hat{n}\hat{n}^T)$$

$\frac{\partial F}{\partial \dot{q}}$ is given by:

$$\frac{\partial F}{\partial \dot{q}} = \begin{pmatrix} -B & B \\ B & -B \end{pmatrix}$$

where $B$ is a symmetric $2 \times 2$ matrix given by:

$$B = \beta \hat{n}\hat{n}^T$$

5.3.7.5 Gravitational Force Jacobian

For two particles in 2D interacting with a gravitational force, the force Jacobian is a symmetric $4 \times 4$ matrix

$$\frac{\partial F}{\partial q} = \begin{pmatrix} K & -K \\ -K & K \end{pmatrix}$$

where $K$ is a symmetric $2 \times 2$ matrix given by

$$K = -\frac{Gm_1m_2}{l^3} (I_d - 3\hat{n}\hat{n}^T) .$$

$\hat{n} = (x_i - x_j) / |x_i - x_j|$ denotes a normalized vector pointing from $x_j$ to $x_i$, $G$ denotes the gravitational constant, $m_1$ and $m_2$ denote the masses of the first and second particle, and $l = |x_i - x_j|$ denotes the distance between the two particles. The gravitational force has no velocity dependence, so $\frac{\partial F}{\partial \dot{q}} = 0$. 
5.3.7.6 Linear Drag Force Jacobian

For each particle, the linear drag force has a $2 \times 2$ force Jacobian of:

$$ \frac{\partial F}{\partial \dot{q}} = -\beta (\text{Id}) $$

The linear drag force has no position dependence, so $\frac{\partial F}{\partial q} = 0$. \text{Id} is the identity matrix.

5.3.8 Fixed Vertices

To fix a vertex, we want an iteration of Newton’s method to produce no change in that vertex’s position or velocity. One way to accomplish this is to set each of the vertex’s degrees of freedom in the right hand side of (5.2) to 0, to set the row and column corresponding to the vertex’s degrees of freedom in the left hand side of (5.2) to 0, and to set the diagonal entry corresponding to the vertex’s degrees of freedom in the left hand side of (5.2) to 1. At the end of an iteration of Newton’s method (just one iteration in the case of linearized implicit Euler), the net result will be that the change in the vertex’s velocity, and thus position, is 0.

For example, say we want to fix vertex 5. We would clear entries 10 and 11 in the right hand side of (5.2). We would also clear rows 10 and 11 and columns 10 and 11 in the left hand side of (5.2). We would finally set entry (10, 10) and entry (11, 11) of the left hand side of (5.2) to 1. When we now solve this linear system, the result will be that the $x$ and $y$ coordinate of vertex 5 will remain unchanged.

5.3.9 Full Implicit Euler (Extra Credit)

Recall that linearized implicit Euler is equivalent to taking a single step of Newton’s method. We can run Newton’s method to convergence if desired, however. One additional complication this introduces is the question of how to know when our system has reached convergence. There are a number of choices. We could monitor the magnitude of the residual, we could monitor the change in the magnitude of the residual between time-steps, we could monitor the magnitude of step size, the options go on.

For our purposes, it will suffice to monitor the absolute magnitude of the step-size. That is, at the end of each iteration of Newton’s method, if $|\dot{q}^{n+1} - \dot{q}^{n+1}| < 1.0^{-9}$, we declare that Newton’s method has converged.
Implement implicit Euler as detailed in \( \text{(5.1)} \) using the source file `ImplicitEuler.cpp`. Note that this feature is NOT required to obtain a 100% on the assignment. The extra credit scenes have been placed in a separate folder, `extracreditassets`. The grading oracle will work with these scenes independently of the required scenes.

### 5.3.10 Vortex Force (Free Feature for Creative Scene)

We have provided a force for you that generates a vortex-like effect. We have also provided an example file, `assets/VortexExample.xml`, that demonstrates the use of this force. You are welcome to use this force in one of your creative scenes, but this is not required.

### 5.4 Collisions: Theme II Milestone I

#### 5.4.1 Introduction

In Theme II, you will add efficient and robust collision handling to the particle system you worked with in Theme I. In this milestone you will implement basic but functional collision detection and response, which will be built upon in Milestones II and III. Test scenes for Theme II are located in a new assets directory, `theme2assets`.

#### 5.4.2 New XML Features

In addition to the xml tags from Theme I, this milestone adds the following new features:

1. The `halfplane` node adds a half-plane to the scene:

   ```xml
   <halfplane px="0.0" py="0.0" nx="0.0" ny="1.0"/>
   ```

   The half-plane consists of all points \( \mathbf{x} \) satisfying \( \mathbf{x} - (px, py) \cdot (nx, ny) \leq 0 \). So the example half-plane node above would consist of a floor taking up all points with \( y \leq 0 \). The vector \( (nx, ny) \) must have nonzero magnitude but need not be normalized; this magnitude does not affect the behavior of the half-plane. This XML tag is associated with the half-plane feature described in section \( \text{5.4.3.5} \).

2. The `halfplanecolor` node changes a half-plane’s color.
The integer attribute \( i \) identifies the half-plane. The scalar \( r \), \( g \), and \( b \) attributes set the half-plane’s color. The three color attributes must have values between 0.0 and 1.0.

3. The \texttt{collision} node specifies how collisions are handled during the simulation:

\[
\texttt{<collision type="simple" COR="0.5"/>}
\]

For this milestone, the valid values for \texttt{type} are “simple” and “none”. Later milestones will add new types of collision handling. The optional attribute \texttt{COR} sets the coefficient of restitution (described below) for contact. Its default value is 1.0.

4. To give you greater control over the rendering of your simulation, we have added an optional \texttt{viewport} node:

\[
\texttt{<viewport cx="0" cy="0" size="5.0"/>}
\]

The viewport specifies the center and size of the default camera viewport, in object coordinates; these settings override the default auto-centering behavior. The camera can still be moved around and resized during an OpenGL simulation using the mouse.

The headless rendering viewport has been completely changed to coincide with what you see during OpenGL rendering, and in particular respects the \texttt{viewport} settings.

### 5.4.3 Required Features for Milestone I

#### 5.4.3.1 Overview of Detection

Suppose you have two objects (each of which can be a particle, edge, or half-plane) \( O_1 \) and \( O_2 \) with thicknesses \( r_1 \) and \( r_2 \). Conceptually, the algorithm for checking whether or not \( O_1 \) and \( O_2 \) are colliding is to

1. look at all vectors between points in \( O_1 \) and \( O_2 \),

2. find the vector \( \mathbf{n} \) that is the shortest,
Figure 5.5: Particle vs. edge collisions: Left: Checking whether or not a particle and an edge are colliding. Several vectors between the particle and and the edge are drawn in blue; the shortest one $n$ is drawn in red. Since $n$ has length greater than the sum of the two objects’ radii, the objects are not overlapping and so are not colliding. Right: Two edges cannot collide without a particle-edge collision also occurring (particle drawn in red).

3. and compare its length to $r_1 + r_2$. If $n$ has length less than $r_1 + r_2$, the objects are overlapping.

4. Lastly, check if they are approaching or moving apart. If they are approaching, then the objects are colliding.

Figure 5.5 left, illustrates this algorithm for a particle and an edge.

Although this algorithm gives a good overview of the big picture, it cannot be practically implemented as written above: for instance, you wouldn’t want to write code to actually find the set of all possible vectors between points in $O_1$ and $O_2$. Instead, by examining the geometry of the colliding objects, we will derive formulas for the shortest vector, skipping the first step entirely.

The types of collisions that can occur are particle–particle, particle–edge, and particle–half-plane. Edges can also collide with edges, and edges can collide against half-planes – but in both of these cases there is also guaranteed to be a particle–edge or particle–half-plane collision as well, so they do not need to be checked or responded to separately. (For this theme you may
5.4.3.2 Implement Detection Routines for Primitive Pairs

Edit the provided source file `SimpleCollisionHandler.cpp` and, for each of the following pairs of primitives, implement a function that takes two such primitives and computes $\mathbf{n}$ and determines whether or not the two objects are colliding.

5.4.3.3 Particle–Particle

Finding the shortest vector from $O_1$ to $O_2$ is trivial when both objects are particles: there is only one vector to choose from. When checking the length of $\mathbf{n}$, don’t forget that the two particles might have different radii.

To check if the particles are approaching, look at the difference in velocity along the $\mathbf{n}$ direction, $(\mathbf{v}_1 - \mathbf{v}_2) \cdot \mathbf{n}$. This scalar is proportional to the speed at which the particles are moving toward each other; if it is positive, then the particles are approaching.

**Important Note:** The oracle uses a *strict* inequality ($<$) when comparing $|\mathbf{n}|$ to $r_1 + r_2$. It also uses a strict inequality when checking if the relative velocity along $\mathbf{n}$ is positive. Be sure you follow this convention – otherwise you may detect collisions that the oracle does not, and fail a test.
5.4.3.4 Particle–Edge

If $O_1$ is a particle and $O_2$ an edge, there are many possible vectors from $O_1$ to $O_2$: although there is only one choice for the vector’s tail, the tip can be any point on the edge $O_2$. To find the shortest vector, you must find the closest point on the edge to the particle.

Let $x_1$ be the position of the particle, and $x_2$ and $x_3$ the two endpoints of the edge. We will first find the closest point to $x_1$ on the infinite line passing through these two endpoints. Note that any point on the line can be written as $x(\alpha) = x_2 + \alpha(x_3 - x_2)$ for some scalar parameter $\alpha$. Our goal is to find the $\alpha$ that gives us a point closest to $x_1$.

Minimizing the distance between $x_1$ and $x(\alpha)$ is the same as minimizing the squared distance, $\|x_1 - x(\alpha)\|^2$, between them. As usual when trying to find a minimum, we take the derivative with respect to $\alpha$, set it equal to zero, then solve for $\alpha$:

$$\frac{d}{d\alpha}\|x_1 - x(\alpha)\|^2 = 0$$
$$\frac{d}{d\alpha}[(x_1 - x(\alpha)) \cdot (x_1 - x(\alpha))] = 0$$
$$-2(x_1 - x(\alpha)) \cdot \frac{d}{d\alpha}x(\alpha) = 0$$
$$(x_1 - x(\alpha)) \cdot \frac{d}{d\alpha}[x_2 + \alpha(x_3 - x_2)] = 0$$
$$(x_1 - x_2 - \alpha(x_3 - x_2)) \cdot (x_3 - x_2) = 0$$
$$(x_1 - x_2) \cdot (x_3 - x_2) = \alpha\|x_3 - x_2\|^2$$
$$\alpha = \frac{(x_1 - x_2) \cdot (x_3 - x_2)}{\|x_3 - x_2\|^2}.$$

Plugging this value of $\alpha$ into $x(\alpha)$ tells us the closest point on the infinite line to $x_1$. What we really want to know is the closest point on the segment to $x_1$. If $0 \leq \alpha \leq 1$, then those points are one and the same. Otherwise, we have to clamp $\alpha$ to the range $[0, 1]$.

The full algorithm for finding $n$ is thus:

1. Calculate $\alpha$ using the formula above.
2. If $\alpha < 0$, set $\alpha = 0$. If $\alpha > 1$, set $\alpha = 1$.
3. Calculate the point $x(\alpha)$.
4. The vector we need is then $n = x(\alpha) - x_1$. 
Implement this algorithm.

To check if the particle is approaching the edge, we want to look at the velocities in the \( \mathbf{n} \) direction of the particle and the closest point on the edge. This closest point is \( \mathbf{x}(\alpha) \); the velocity of this point is given by taking the velocities of the endpoints and interpolating using the same \( \alpha \):

\[
\mathbf{v}(\alpha) = \mathbf{v}_2 + \alpha (\mathbf{v}_3 - \mathbf{v}_2).
\]

The difference in velocity along \( \mathbf{n} \) is then proportional to \((\mathbf{v}_1 - \mathbf{v}(\alpha)) \cdot \mathbf{n}\). Once again, the objects are approaching if this scalar is positive.

**5.4.3.5 Particle–Half-plane**

Let \( \mathbf{x}_h \) and \( \mathbf{n}_h \) be the position and normal of the half-plane, as specified in the XML file. The shortest vector between the particle and half-plane has direction \( -\mathbf{n}_h \) and magnitude equal to the distance between the particle and boundary of the half-plane. This distance is equal to

\[
\frac{(\mathbf{x}_h - \mathbf{x}) \cdot -\mathbf{n}_h}{\|\mathbf{n}_h\|},
\]

so

\[
\mathbf{n} = \frac{(\mathbf{x}_h - \mathbf{x}) \cdot \mathbf{n}_h}{\|\mathbf{n}_h\|^2} \mathbf{n}_h.
\]

The radius of the half-plane is zero for the purposes of checking the length of \( \mathbf{n} \).

The velocity of the particle in the direction \( \mathbf{n} \) is proportional to \( \mathbf{v} \cdot \mathbf{n} \). If this quantity is positive, the particle is approaching the half-plane.

**5.4.3.6 Automatic Testing of Detection by the Oracle**

To aid you in debugging your code, we have added functionality to the oracle to automatically compare the collisions you found against those the oracle is expecting. When running the oracle in binary input mode, during each frame it will show you:

1. Missed collisions: those that the oracle detected this frame, but weren’t detected by your code. The shortest vector \( \mathbf{n} \) between the two objects whose collision was missed is drawn in green to indicate the missed collision.
2. Superfluous collisions: those that your code detected, but that the oracle didn’t. The
tool 

vector \( \mathbf{n} \) of the superfluous collision is drawn on top of the simulation in red.

3. Incorrect \( \mathbf{n} \): both the oracle and your code agree that a collision occurred, but disagree
on the shortest vector \( \mathbf{n} \). The incorrect vector is drawn in red, and the correct one in

green.

### 5.4.3.7 Overview of Contact Response

Once we detect a collision, we must apply \textit{impulses} – instantaneous changes to momentum – to

\textit{respond} to the collision. Edit the provided source file \textit{SimpleCollisionHandler.cpp} to correctly

respond to each of the three types of possible collisions.

### 5.4.3.8 Particle–Particle

Denoting post-collision-response velocities with tildes, we apply an impulse to the velocities of

the particles \( O_1 \) and \( O_2 \) using the equations

\[
\begin{align*}
\tilde{v}_1 & \leftarrow v_1 + I_1/m_1 \\
\tilde{v}_2 & \leftarrow v_2 + I_2/m_2
\end{align*}
\]

for some as-yet-undetermined impulse vectors \( I_1 \) and \( I_2 \). By choosing these impulse vectors
carefully, we stop the particles from approaching any further, while obeying the laws of physics.

Denote by \( \hat{\mathbf{n}} \) the unit vector in direction \( \mathbf{n} \). \( \hat{\mathbf{n}} \) is called the \textit{contact normal}. Pushing the

objects apart in the direction of this vector most quickly increases the distance between them,

so we want the impulses \( I_1 \) and \( I_2 \) to lie in the same direction as \( \hat{\mathbf{n}} \):

\[
\begin{align*}
I_1 &= I_1 \hat{\mathbf{n}} \\
I_2 &= I_2 \hat{\mathbf{n}}
\end{align*}
\]

for scalars \( I_1 \) and \( I_2 \).

To find formulas for these scalars, we write down the laws of conservation of momentum

and energy. Conservation of momentum tells us that the total momentum of the two objects
before and after we apply the impulses should be equal. Therefore

\[ m_1 \mathbf{v}_1 + m_2 \mathbf{v}_2 = m_1 \tilde{\mathbf{v}}_1 + m_2 \tilde{\mathbf{v}}_2 \]

\[ m_1 (\mathbf{v}_1 - \mathbf{v}_1 - I_1/m_1) = m_2 (\mathbf{v}_2 + I_2/m_2 - \mathbf{v}_2) \]

\[ -I_1 = I_2 \]

\[ -I_1 \hat{n} = I_2 \hat{n} \]

\[ -I_1 \cdot \hat{n} = I_2 \hat{n} \cdot \hat{n} \]

\[ -I_1 = I_2. \]

Similarly, energy before and after applying the impulse must be equal. In particular, since applying an impulse only modifies velocities and not positions, and so doesn’t change potential energy, kinetic energy before and after the impulse must be equal:

\[ \frac{1}{2} m_1 \mathbf{v}_1 \cdot \mathbf{v}_1 + \frac{1}{2} m_2 \mathbf{v}_2 \cdot \mathbf{v}_2 = \frac{1}{2} m_1 \tilde{\mathbf{v}}_1 \cdot \tilde{\mathbf{v}}_1 + \frac{1}{2} m_2 \tilde{\mathbf{v}}_2 \cdot \tilde{\mathbf{v}}_2 \]

\[ m_1 \mathbf{v}_1 \cdot \mathbf{v}_1 + m_2 \mathbf{v}_2 \cdot \mathbf{v}_2 = m_1 \mathbf{v}_1 \cdot \mathbf{v}_1 + 2I_1 \mathbf{v}_1 \cdot \hat{n} + I_1^2/m_1 \hat{n} \cdot \hat{n} \]

\[ + m_2 \mathbf{v}_2 \cdot \mathbf{v}_2 + 2I_2 \mathbf{v}_2 \cdot \hat{n} + I_2^2/m_2 \hat{n} \cdot \hat{n} \]

\[ 0 = 2I_1 \mathbf{v}_1 \cdot \hat{n} + 2I_2 \mathbf{v}_2 \cdot \hat{n} + I_1^2/m_1 + I_2^2/m_2. \]

Since we know \( I_2 = -I_1 \),

\[ 0 = 2I_1 \mathbf{v}_1 \cdot \hat{n} - 2I_1 \mathbf{v}_2 \cdot \hat{n} + I_1^2/m_1 + I_1^2/m_2 \]

\[ 0 = 2(\mathbf{v}_1 - \mathbf{v}_2) \cdot \hat{n} + I_1/m_1 + I_1/m_2 \]

\[ I_1(m_1 + m_2) = 2m_1 m_2 (\mathbf{v}_2 - \mathbf{v}_1) \cdot \hat{n} \]

\[ I_1 = \frac{2(\mathbf{v}_2 - \mathbf{v}_1) \cdot \hat{n}}{\frac{1}{m_1} + \frac{1}{m_2}}. \]

Therefore the final formulas for updating velocities when a collision is detected are

\[ \tilde{\mathbf{v}}_1 \leftarrow \mathbf{v}_1 + \frac{2(\mathbf{v}_2 - \mathbf{v}_1) \cdot \hat{n}}{\frac{1}{m_1} + \frac{1}{m_2}} \hat{n} \]

\[ \tilde{\mathbf{v}}_2 \leftarrow \mathbf{v}_2 - \frac{2(\mathbf{v}_2 - \mathbf{v}_1) \cdot \hat{n}}{\frac{m_2}{m_1} + 1} \hat{n}. \]

5.4.3.9 Particle–Edge

When a particle hits an edge, we again want to apply an impulse to the particles associated to that particle and edge. Unlike the particle-particle case, there are three particles involved in a
particle-edge collision: the colliding particle $x_1$, and the two endpoints of the colliding edge $x_2$ and $x_3$. As before, we want to apply an impulse in the direction $\mathbf{n}$ to each of these particles:

\[
\tilde{v}_1 \leftarrow v_1 + I_1/m_1 \mathbf{n} \\
\tilde{v}_2 \leftarrow v_2 + I_2/m_2 \mathbf{n} \\
\tilde{v}_3 \leftarrow v_3 + I_3/m_3 \mathbf{n}.
\]

We need to solve for the values of the unknown scalars $I_1$, $I_2$, and $I_3$ by applying conservation laws.

Let $x_e$ be the closest point on the edge to the particle. We know we can write this point as $x_2 + \alpha(x_3 - x_2)$ for a scalar $\alpha$ (see the section on particle-edge detection for the formula for $\alpha$.) Rewriting this equation yields

\[
x_e = (1 - \alpha)x_2 + \alpha x_3.
\]

We can now look at the angular momentum of the particle and edge about the point $x_e$; we want it to be the same before and after the impulse is applied:

\[
m_1 v_1 \times (x_1 - x_e) + m_2 v_2 \times (x_2 - x_e) + m_3 v_3 \times (x_3 - x_e) = \\
m_1 \tilde{v}_1 \times (x_1 - x_e) + m_2 \tilde{v}_2 \times (x_2 - x_e) + m_3 \tilde{v}_3 \times (x_3 - x_e).
\]

Substituting for the updated velocities and canceling terms from both sides, we get

\[
0 = I_1 \mathbf{n} \times (x_1 - x_e) + I_2 \mathbf{n} \times (x_2 - x_e) + I_3 \mathbf{n} \times (x_3 - x_e).
\]

We know that $x_1 - x_e$ lies in the same direction as $\mathbf{n}$: in fact, the latter is the normalized version of the former, by definition. Therefore $\mathbf{n} \times (x_1 - x_e) = 0$ and

\[
0 = I_2 \mathbf{n} \times (x_2 - x_e) + I_3 \mathbf{n} \times (x_3 - x_e)
\]

\[
0 = I_2 \mathbf{n} \times (x_2 - (1 - \alpha)x_2 - \alpha x_3) + I_3 \mathbf{n} \times (x_3 - (1 - \alpha)x_2 - \alpha x_3)
\]

\[
0 = I_2 \mathbf{n} \times \alpha(x_2 - x_3) - I_3 \mathbf{n} \times (1 - \alpha)(x_2 - x_3)
\]

\[
0 = (\alpha I_2 - (1 - \alpha)I_3) \mathbf{n} \times (x_2 - x_3).
\]

If we assume that $\mathbf{n} \times (x_2 - x_3)$ is not the zero vector (this assumption is valid except in the corner case where the particle is travelling along the infinite line coincident to the edge), we
must have that the coefficient is 0:

\[
\alpha I_2 - (1 - \alpha)I_3 = 0 \\
I_2 = \frac{1 - \alpha}{\alpha} I_3.
\]

If we define the new quantity \( I_e = \frac{I_2}{\alpha} \), we thus have

\[
\begin{align*}
\tilde{v}_1 & \leftarrow v_1 + (I_1/m_1) \hat{n} \\
\tilde{v}_2 & \leftarrow v_2 + ((1 - \alpha)I_e/m_2) \hat{n} \\
\tilde{v}_3 & \leftarrow v_3 + (\alpha I_e/m_3) \hat{n},
\end{align*}
\]

and only two unknowns remain.

The impulse must also conserve linear momentum. Therefore

\[
m_1 v_1 + m_2 v_2 + m_3 v_3 = m_1 \tilde{v}_1 + m_2 \tilde{v}_2 + m_3 \tilde{v}_3.
\]

Substituting and simplifying yields

\[
\begin{align*}
0 &= (I_1 + (1 - \alpha)I_e + \alpha I_e) \hat{n} \\
0 &= I_1 + (1 - \alpha)I_e + \alpha I_e \\
0 &= I_1 + I_e \\
I_e &= -I_1.
\end{align*}
\]

Lastly, we have conservation of energy:

\[
\begin{align*}
\frac{1}{2} m_1 v_1 \cdot v_1 + \frac{1}{2} m_2 v_2 \cdot v_2 + \frac{1}{2} m_3 v_3 \cdot v_3 &= \frac{1}{2} m_1 \tilde{v}_1 \cdot \tilde{v}_1 + \frac{1}{2} m_2 \tilde{v}_2 \cdot \tilde{v}_2 + \frac{1}{2} m_3 \tilde{v}_3 \cdot \tilde{v}_3 \\
-2I_1 v_1 \cdot \hat{n} + 2(1 - \alpha)I_1 v_2 \cdot \hat{n} + 2\alpha I_1 v_3 \cdot \hat{n} &= (I_1^2/m_1 + (1 - \alpha)^2I_2^2/m_2 + \alpha^2I_3^2/m_3) \hat{n} \cdot \hat{n} \\
-2v_1 \cdot \hat{n} + 2v_e \cdot \hat{n} &= I_1(1/m_1 + (1 - \alpha)^2/m_2 + \alpha^2/m_3) \\
\frac{2(v_e - v_1) \cdot \hat{n}}{1/m_1 + (1 - \alpha)^2/m_2 + \alpha^2/m_3} &= I_1,
\end{align*}
\]

where

\[
v_e = (1 - \alpha)v_2 + \alpha v_3.
\]
The final formulas for the changes in velocity are thus

\[
\begin{align*}
\tilde{v}_1 &\leftarrow v_1 + \frac{2(v_e - v_1) \cdot \hat{n}}{1 + \frac{(1-\alpha)^2 m_1}{m_2} + \frac{\alpha^2 m_1}{m_3}} \hat{n} \\
\tilde{v}_2 &\leftarrow v_2 - \frac{2(1-\alpha)(v_e - v_1) \cdot \hat{n}}{m_2} \hat{n} + \frac{(1-\alpha)^2}{m_2} + \frac{\alpha^2 m_2}{m_3} \hat{n} \\
\tilde{v}_3 &\leftarrow v_3 - \frac{2\alpha(v_e - v_1) \cdot \hat{n}}{m_3} \hat{n} \cdot \hat{n} + \frac{(1-\alpha)^2 m_3}{m_2} + \alpha^2 \hat{n},
\end{align*}
\]

(5.4)

Before implementing these formulas, it may be helpful for you to think about the case when \(x_e\) is one of the edge’s two endpoints, and verify that the formulas above reduce to the particle–particle case.

5.4.3.10 Particle–Half-Plane

Since the half-plane is fixed in place, we only have a particle to which we must apply an impulse:

\[
\tilde{v} \leftarrow v + \frac{I}{m} \hat{n}.
\]

Conservation of energy tells us the right value of \(I\):

\[
\frac{1}{2} m\tilde{v} \cdot \tilde{v} = \frac{1}{2} m v \cdot v
\]

\[
2I/m \cdot \hat{n} + \frac{I^2}{m^2} \hat{n} \cdot \hat{n} = 0
\]

\[
I/m^2 = -2/m \cdot v \cdot \hat{n}
\]

\[
I = -2m \cdot v \cdot \hat{n}.
\]

The final formula for the change in the velocity of the particle is thus

\[
\tilde{v} \leftarrow v - 2(v \cdot \hat{n})\hat{n}.
\]

5.4.3.11 Fixed Particles

Fixed particles and edges require special consideration. On the one hand, we don’t want to apply any impulses to them that will change their velocity, since they are supposed to be fixed in place. On the other, we can’t ignore them completely during collision detection and response, since we do want non-fixed particles and edges to interact with them.
Since changes in velocity are equal to impulse divided by mass, setting the mass of fixed particles to infinity is one way of ensuring that they are “immovable objects” during collisions. Implement this handling of fixed objects by doing the following:

- Instead of applying an impulse to a fixed particle, do nothing instead. (This step is necessary to avoid $\frac{\infty}{\infty}$ in certain degenerate cases.)

- Any time you would use the mass of a fixed particle while applying an impulse to a non-fixed particle, use infinity for that mass instead.

**Coding tip:** In C++, calling `std::numeric_limits<double>::infinity()` generates infinity.

**Important note:** Ensure that you have implemented the formulas for applying impulses (equations 5.3 and 5.4) exactly as they are written in this PDF. They are specifically formulated to work correctly when plugging in infinite masses. It is possible to rewrite these formulas in ways that are mathematically equivalent for finite masses, but give $\frac{\infty}{\infty}$ when fixed particles are involved.

### 5.4.3.12 Coefficient of Restitution

When you drop a ball on the ground, conservation of energy suggests that the ball must bounce back up to exactly its original height. Of course, balls in the real world do not do this: bowling balls barely bounce up at all, and even the bounciest rubber ball only returns to a fraction of its original height.

Many complicated factors contribute to this dissipation of energy: plastic deformation, internal friction during elastic deformation, and production of sound waves are just a few examples. Simulating all of these small-scale phenomena is not feasible, so instead many simulations approximate the net coarse-scale behavior of these effects by a coefficient of restitution (COR). The COR is the ratio of the maximum height of an object after a bounce to the height before the bounce: for a bowling ball it is near 0, and for a rubber ball, almost 1.

Although the derivation is beyond the scope of this course, it turns out that COR can be simulated by multiplying all impulses applied during contact response by the following scalar:

$$\frac{1 + \text{COR}}{2}.$$
Notice that when COR is 1.0, the impulses don’t change; this is as expected, since COR=1.0 corresponds to perfect conservation of energy.

Implement COR.

5.5 Collisions: Theme II Milestone II

5.5.1 Introduction

In Theme II, we will add efficient and robust collision handling to the particle system we developed in Theme I. In this milestone we will improve on the collision detection we implemented in Milestone I. We will extend the detection to be continuous in time to prevent objects from tunneling straight through each other or becoming embedded in one another. We will also explore a completely different way of handling collisions, the penalty method. Test scenes for this milestone are located in a new directory, theme2assets/Milestone02.

5.5.2 New XML Features

This milestone adds the following new feature:

1. The collision node’s functionality has been expanded:

   \[
   <\text{collision} \text{ type}="\text{penalty}" \text{ k}"=100" \text{ thickness}="0.1"/> \]

   For this milestone, the valid values for type are “none”, “simple”, “continuous-time”, and “penalty”; you will be implementing the latter two in this milestone. In addition to the COR attribute described in Milestone I, collision has new optional attributes k and thickness specifying the stiffness and thickness of the penalty forces used by the penalty method (described below). These attributes have no effect unless type is “penalty”.

5.5.3 Required Features for Milestone II

5.5.3.1 Dealing With Tunneling

In Milestone I we implemented very simple collision detection: at the end of every simulation step, we calculated the distances between objects to check if they were overlapping; if so, we
checked their relative velocity in the normal direction to see if they were approaching. If two objects were both overlapping and approaching, a collision was said to have occurred.

Although this algorithm will do in a pinch when objects move slowly and/or the simulation time-step is small, it is not very robust. In particular, it is possible for objects to move far enough during a single time-step to tunnel completely through a second object; this collision cannot be detected without taking into account the motion of the particle throughout the entire time-step, and not just its position at the end of the time-step. Figure 5.7 illustrates this tunneling.

Suppose we have the positions of objects at the start of a time-step ($q^s$) and the end of the time-step ($q^e$). We assume that the particles and edges moved in straight lines between the two time-steps; in other words, that the positions between the two time-steps are given by the continuous function $q(t) = q^s + t\Delta q$, where $\Delta q = q^e - q^s$. The parameter $t$ can be thought of as the time since the start of the time-step, where for convenience time has been renormalized so that the time-step has duration 1.

To prevent tunneling, we must ask ourselves, “is there any time $t$ in $[0, 1]$ when the two objects are (a) overlapping, and (b) approaching?” Answering this question is called performing continuous-time collision detection, since we are looking at all values of $t$ instead of
just \( t = 1 \).

For each pair of objects \( O_1, O_2 \) (with radii \( r_1 \) and \( r_2 \)) we derived in Milestone I formulas for the shortest vector \( \mathbf{n} \) between them. These formulas depended only on the end-of-time-step positions \( \mathbf{x}^e \) of the objects; we can thus write down the shortest vector \( \mathbf{n}(t) \) as a function of time by replacing \( \mathbf{x}^e \) with \( \mathbf{x}(t) = \mathbf{x}^e + t\Delta\mathbf{x} \) in these formulas. Question (a) then boils down to, “is there some time \( t \) in \([0, 1]\) when \( |\mathbf{n}(t)| < r_1 + r_2 \)?” Similarly, question (b) can be rephrased as, “is there some time \( t \) in \([0, 1]\) when the relative velocity \( \mathbf{r}(\Delta\mathbf{x}) \) of the two objects along \( \mathbf{n}(t) \) is positive?”

Performing continuous-time collision detection is thus equivalent to checking whether the simultaneous system of inequalities

\[
\sqrt{\mathbf{n}(t) \cdot \mathbf{n}(t)} < r_1 + r_2 \\
\mathbf{r}(\Delta\mathbf{x}) \cdot \mathbf{n}(t) > 0
\]

has a solution between \( t = 0 \) and \( t = 1 \). Solving such systems of inequalities is in general very hard. One thing we can do to make our lives a little bit easier is to square both sides of the first inequality, getting rid of the square root; this squaring is permitted since both sides of the inequality must be positive.

\[
\mathbf{n}(t) \cdot \mathbf{n}(t) < (r_1 + r_2)^2 \\
\Delta\mathbf{x} \cdot \mathbf{n}(t) > 0.
\]

Later in these milestone instructions we will see that for particle–particle, particle–edge, and particle–halfplane collisions, these equations can be written as polynomial inequalities, which are much easier to solve than general non-linear inequalities.

### 5.5.3.2 Systems of Polynomial Inequalities

Suppose we have some polynomials \( f(t), g(t) \) and we want to find a simultaneous solution to \( f(t) \geq 0 \), \( g(t) \geq 0 \).
Figure 5.8: Systems of inequalities: Solving the system of inequalities $f(t) > 0$, $g(t) > 0$ for two example polynomials. Left: Plots of the two polynomials. Right: The intervals of $t$ on which $f$ and $g$ are positive, and the intersection (bottom, in red) of the two sets of intervals. Any point inside a red interval is a solution to the system of inequalities.

We take one of our polynomials, $f$, and find all of its roots. We know that between consecutive roots, $f$ must have the same sign (positive or negative), since $f$’s sign can only change at a root. By evaluating $f(t)$ at any value of $t$ between the two roots we can determine this sign.

We can thus find the intervals of $t$ on which $f(t)$ is positive. For example, the polynomial $x^2 - 1$ has two intervals on which it is positive: $(-\infty, -1)$, and $(1, \infty)$. $-x^2 + 1$ has one: $(-1, 1)$. Higher-degree polynomials might have many such intervals.

We can then do the same thing for $g(t)$. If any of $f$’s intervals overlap one of $g$’s intervals, then any value of $t$ inside the intersection of the two intervals is a solution to the system of inequalities. Figure 5.8 illustrates this algorithm for two example polynomials.

As a final note, although this algorithm has been described for two polynomials, it is easy to extend to the problem of solving for when an arbitrary number of polynomials $f(t), g(t), h(t), \ldots$ are simultaneously positive.

This algorithm, while conceptually simple, can be tricky to implement correctly, so in ContinuousTimeCollisionHandler.cpp we have provided you with the method findFirstIntersection-Time that takes in an arbitrary number of polynomials and returns the first time $t > 0$ when
all polynomials become simultaneously positive.

**Important Note:** Carefully read the comments in the provided code, and ensure you are e.g. passing in polynomial coefficients in the right order.

### 5.5.3.3 Incorporating Continuous-Time Collision Detection with Collision Response

Using the above polynomial inequality algorithm, we can determine whether or not two objects collide during a time-step. But what do we do with this information? In Milestone I, we applied impulses at the end of the time-step, but there’s no point detecting that two objects are going to tunnel through each other if we can’t do anything about it until after the tunneling has occurred!

What we must do instead is partition time-stepping into two parts. First, we step the simulation forward using whatever integrator has been specified for the simulation, completely ignoring collisions. This gives us predicted positions and velocities $q^e$ and $\dot{q}^e$. We then perform continuous-time collision detection and response, modifying $q^e$ and $\dot{q}^e$ to prevent all collision during the time-step:

1. Step the old positions $q^s$ and velocities $\dot{q}^s$ forward using the numerical integrator to get predicted new positions $q^e$ and velocities $\dot{q}^e$.

2. **Assume** the particles moved with constant velocity from $q^s$ to $q^e$, and **perform** continuous-time collision detection with $q(t) = q^s + t\Delta q$, where $\Delta q = q^e - q^s$.

3. If any collisions were detected, **apply impulses** to the predicted velocities $\dot{q}^e$ to get modified velocities $\dot{q}^m = \dot{q}^s + M^{-1}I$. The vector $I$ is the impulse derived in Milestone I needed to resolve the collision.

4. **Also modify positions** to get new collision-free positions: $q^m = q^e + hM^{-1}I$.

Note that this algorithm assumes that applying impulses does not cause any new collisions. In Milestone III we will examine this assumption and further improve collision response.

Your first task in this milestone is to modify `ContinuousTimeCollisionHandler.cpp` and implement step 2 of this algorithm. For each of particle–particle, particle–edge, and particle–half-plane, implement continuous-time collision detection: given old and new positions, and
assuming objects move in a straight line from their old to new positions, determine whether or not the objects are overlapping and approaching at some point during the motion. If so, compute the value of \( t \) at which the collision occurs, and the shortest vector between the objects at that value of \( t \).

5.5.3.4 Particle–Particle

For two particles with centers \( x_1(t) \) and \( x_2(t) \), the vector \( n(t) \) is simply \( x_2(t) - x_1(t) \). The first polynomial we need is therefore

\[
\mathbf{n} \cdot \mathbf{n} < (r_1 + r_2)^2
\]

\[
(x_2^s + t\Delta x_2 - x_1^s - t\Delta x_1) \cdot (x_2^s + t\Delta x_2 - x_1^s - t\Delta x_1) < (r_1 + r_2)^2
\]

\[
[(x_2^s - x_1^s) + t(\Delta x_2 - \Delta x_1)] \cdot [(x_2^s - x_1^s) + t(\Delta x_2 - \Delta x_1)] < (r_1 + r_2)^2
\]

\[
(x_2^s - x_1^s) \cdot (x_2^s - x_1^s) + 2t(x_2^s - x_1^s) \cdot (\Delta x_2 - \Delta x_1) + t^2(\Delta x_2 - \Delta x_1) \cdot (\Delta x_2 - \Delta x_1) < (r_1 + r_2)^2,
\]

a quadratic in \( t \). We need to rewrite this in the form \( f(t) > 0 \) in order to plug it into the polynomial inequality solver:

\[
-(\Delta x_2 - \Delta x_1) \cdot (\Delta x_2 - \Delta x_1)t^2 - 2(x_2^s - x_1^s) \cdot (\Delta x_2 - \Delta x_1)t + (r_1 + r_2)^2 - (x_2^s - x_1^s) \cdot (x_2^s - x_1^s) > 0.
\]

From Milestone I we know that the relative velocity \( \mathbf{r}(\Delta x) \) is \( \Delta x_1 - \Delta x_2 \), so the second inequality is just

\[
(\Delta x_1 - \Delta x_2) \cdot \mathbf{n}(t) > 0
\]

\[
(\Delta x_1 - \Delta x_2) \cdot (x_2^s + t\Delta x_2 - x_1^s - t\Delta x_1) > 0
\]

\[
(\Delta x_1 - \Delta x_2) \cdot [(x_2^s - x_1^s) + t(\Delta x_2 - \Delta x_1)] > 0
\]

\[
(\Delta x_1 - \Delta x_2) \cdot (\Delta x_2 - \Delta x_1)t + (\Delta x_1 - \Delta x_2) \cdot (x_2^s - x_1^s) > 0.
\]

If both polynomial inequalities are satisfied for some \( t \) with \( 0 \leq t \leq 1 \), then a collision has occurred this time-step.
5.5.3.5 Particle–Edge

Recall that in Milestone I, we computed the shortest vector $\mathbf{n}$ between a particle at $\mathbf{x}_1$ with radius $r_v$ and an edge with endpoints at $\mathbf{x}_2$ and $\mathbf{x}_3$ and radius $r_e$ by first extending the edge to an infinite line, and finding the closest point $\mathbf{x}_\alpha = \mathbf{x}_2 + \alpha(\mathbf{x}_3 - \mathbf{x}_2)$ on that line to $\mathbf{x}_1$ by calculating $\alpha$. By clamping $\alpha$ to the range $[0, 1]$, we found the closest point on the segment to $\mathbf{x}_1$, from which it followed that $\mathbf{n} = \mathbf{x}_\alpha - \mathbf{x}_1$.

Because of this clamping, it is not possible to translate the Milestone I formula for $\mathbf{n}$ into a simple function $\mathbf{n}(t)$. Instead we will have to be a little clever: instead of trying to write down a single polynomial inequality encoding that the particle overlaps with the edge, we will instead specify three simultaneous inequalities: (a) the particle overlaps with the infinite line; (b) $\alpha(t) > 0$, and (c) $\alpha(t) < 1$. (For this milestone we won’t worry about collisions with the edge end caps, when $\alpha = 0$ or $\alpha = 1$. Assuming the edge endpoints have radii at least as large as the edge radius, such collisions will be caught by particle–particle collision detection.)

To formulate polynomial (a), we take the formulas from Milestone I and simply omit the clamping of $\alpha$:

$$\mathbf{n}(t) \cdot \mathbf{n}(t) < (r_v + r_e)^2$$

$$(r_v + r_e)^2 - [\mathbf{x}_\alpha(t) - \mathbf{x}_1(t)] \cdot [\mathbf{x}_\alpha(t) - \mathbf{x}_1(t)] > 0$$

$$(r_v + r_e)^2$$

$$- [\mathbf{x}_2(t) + \alpha(t)[\mathbf{x}_3(t) - \mathbf{x}_2(t)] - \mathbf{x}_1(t)] \cdot [\mathbf{x}_2(t) + \alpha(t)[\mathbf{x}_3(t) - \mathbf{x}_2(t)] - \mathbf{x}_1(t)] > 0,$$

where

$$\alpha(t) = \frac{[\mathbf{x}_1(t) - \mathbf{x}_2(t)] \cdot [\mathbf{x}_3(t) - \mathbf{x}_2(t)]}{\|\mathbf{x}_3(t) - \mathbf{x}_2(t)\|^2}.$$  

Unfortunately the left-hand side of this inequality is not a polynomial, since $\alpha(t)$ is a rational function. But we can multiply both sides of the inequality by the (positive) denominator $\|\mathbf{x}_3(t) - \mathbf{x}_2(t)\|^2$, yielding

$$\|\mathbf{x}_3(t) - \mathbf{x}_2(t)\|^2 \left( (r_v + r_e)^2 - \|\mathbf{x}_2(t) + \alpha(t)[\mathbf{x}_3(t) - \mathbf{x}_2(t)] - \mathbf{x}_1(t)\|^2 \right) > 0,$$

which after simplification does turn out to be a quartic polynomial in $t$. Simplifying inequality (a) and deriving the formula for the coefficients of the quartic polynomial is left to you as part
of this assignment. You do not need to turn in written work; whether or not your expressions are correct will be obvious from the behavior of your code. To help with debugging, here is the correct polynomial for two example sets of old positions and change in positions:

1.

\[
x_1^s = (3, 1) \quad \Delta x_1 = (4, 1)
\]
\[
x_2^s = (5, 9) \quad \Delta x_2 = (2, 6)
\]
\[
x_3^s = (5, 3) \quad \Delta x_3 = (5, 8)
\]
\[
r_v = 1 \quad r_e = 1
\]
\[
-361t^4 - 304t^3 - 468t^2 - 288t > 0,
\]

2.

\[
x_1^s = (1, 4) \quad \Delta x_1 = (1, 4)
\]
\[
x_2^s = (2, 1) \quad \Delta x_2 = (3, 5)
\]
\[
x_3^s = (6, 2) \quad \Delta x_3 = (3, 7)
\]
\[
r_v = 1 \quad r_e = 2
\]
\[
-16t^4 - 68t^2 + 36t - 16 > 0.
\]

Polynomial (b) is quadratic in \( t \):

\[
\alpha(t) > 0
\]
\[
\frac{[x_1(t) - x_2(t)] \cdot [x_3(t) - x_2(t)]}{\|x_3(t) - x_2(t)\|^2} > 0
\]
\[
[x_1^s + t\Delta x_1 - x_2^s - t\Delta x_2] \cdot [x_3^s + t\Delta x_3 - x_2^s - t\Delta x_2] > 0
\]
\[
[(x_1^s - x_2^s) + t(\Delta x_1 - \Delta x_2)] \cdot [(x_3^s - x_2^s) + t(\Delta x_3 - \Delta x_2)] > 0
\]
\[
(\Delta x_1 - \Delta x_2) \cdot (\Delta x_3 - \Delta x_2) t^2 + [(x_1^s - x_2^s) \cdot (\Delta x_3 - \Delta x_2) + (\Delta x_1 - \Delta x_2) \cdot (x_3^s - x_2^s)] t
\]
\[
+(x_1^s - x_2^s) \cdot (x_3^s - x_2^s) > 0.
\]

Polynomial (c) is similar, and simplifies to

\[
(\Delta x_1 - \Delta x_3) \cdot (\Delta x_2 - \Delta x_3) t^2 + [(x_1^s - x_3^s) \cdot (\Delta x_2 - \Delta x_3) + (\Delta x_1 - \Delta x_3) \cdot (x_2^s - x_3^s)] t
\]
\[
+(x_1^s - x_3^s) \cdot (x_2^s - x_3^s) > 0.
\]
Figure 5.9: Effect of an approximate $\alpha$: A fast-moving particle collides with an edge at a glancing angle. If we assume $\alpha$ is constant, the dot product between $\mathbf{n}(t)$ (red arrow) and $\Delta \mathbf{x}$ (black arrow) is negative at the time of collision, and we (incorrectly) conclude that the particle and edge are separating at that time.

The last polynomial we need is one encoding that the edge and particle are approaching. Per Milestone I, the full formula for this polynomial would be

$$
(\Delta \mathbf{x}_1 - \Delta \mathbf{x}_2 - \alpha(t)[\Delta \mathbf{x}_3 - \Delta \mathbf{x}_2]) \cdot \mathbf{n}(t) > 0,
$$

which after simplification and clearing out denominators becomes a quintic polynomial. In this milestone we will make the simplifying assumption that $\alpha(t)$ is constant in this inequality. It is important to realize that this approximation is not valid in all cases: if the particle is moving very quickly and collides with an edge at a glancing angle, using this approximation can cause collisions to be missed. Figure 5.9 illustrates one such situation for a moving particle and fixed edge. In production code where tunneling must be avoided at all costs, we would want to use the full quintic polynomial. For this milestone, however, we sacrifice some robustness to avoid gratuitous coding, and instead of the full quintic will use the greatly simplified linear polynomial:

$$
(\Delta \mathbf{x}_1 - \Delta \mathbf{x}_2 - \alpha(0)[\Delta \mathbf{x}_3 - \Delta \mathbf{x}_2]) \cdot (\Delta \mathbf{x}_2 - \Delta \mathbf{x}_1 + \alpha(0)[\Delta \mathbf{x}_3 - \Delta \mathbf{x}_2])t \\
(\Delta \mathbf{x}_1 - \Delta \mathbf{x}_2 - \alpha(0)[\Delta \mathbf{x}_3 - \Delta \mathbf{x}_2]) \cdot (\mathbf{x}_s^2 + \mathbf{x}_s^3 + \alpha(0)[\mathbf{x}_s^2 + \mathbf{x}_s^3 + \Delta \mathbf{x}_3 - \mathbf{x}_1 - \Delta \mathbf{x}_1]) > 0
$$

If all four polynomial inequalities are simultaneously satisfied for some $t$ between 0 and 1,
Figure 5.10: Penalty methods: A conceptual illustration of the penalty method. A spring-like penalty force repels $O_1$ and $O_2$, since they are interpenetrating, and $O_1$ and $O_3$, since they are sufficiently close to each other (less than distance $T$ apart, where $T$ is the penalty method thickness). No force is exerted between $O_2$ and $O_3$ because they are far apart.

then the particle and edge collide.

5.5.3.6 Particle–Half-plane

Since particle–half-plane collisions involve only two degrees of freedom (the position of the particle), its formulas are the simplest to derive. Using the particle–particle derivation above and the formulas from Milestone I as your guide, formulate the two polynomials you need for continuous-time collision detection on your own. Your polynomial encoding that the particle overlaps the half-plane should be quadratic. The one encoding that they are approaching should be linear.

5.5.3.7 Penalty Method

For the second part of Milestone II, you will switch gears and explore a completely different approach to handling collision: the penalty method. The main idea behind the penalty method is simple: for every pair of objects in the simulation, we add a new force – a penalty force – that does nothing if the objects are far apart. If the objects are close or colliding, on the other hand, the force acts like a spring, pushing them apart. The closer the two objects get, the stronger the force becomes, and the harder the simulation tries to push them apart. Figure 5.10 illustrates the penalty method conceptually.
For two objects \(O_1\) and \(O_2\) of radius \(r_1\) and \(r_2\), their penalty potential is given by the formula

\[
V = \begin{cases} 
0, & \|\mathbf{n}\| > r_1 + r_2 + T \\
\frac{1}{2}k(\|\mathbf{n}\| - r_1 - r_2 - T)^2, & \|\mathbf{n}\| \leq r_1 + r_2 + T,
\end{cases}
\]  

(5.5)

where \(k\) is a global penalty force stiffness and \(T\) a global penalty force thickness (set in the scene XML file), and \(\mathbf{n}\) is the usual shortest vector between \(O_1\) and \(O_2\). Notice that this potential bears a lot of similarity to that for a spring, given in Theme I Milestone II, with two notable differences: first, the potential is constant (0) whenever \(\|\mathbf{n}\| > r_1 + r_2 + T\), which holds whenever the objects are further than distance \(T\) apart. Since force is the gradient of potential, a constant potential when the objects aren’t penetrating translates into zero force when the objects are far apart. Second, the length of the spring has been replaced with the length of the shortest vector between the objects, and the rest length has been replaced with the sum of the objects’ radii plus \(T\).

The penalty method has several pros and cons when compared to detecting and applying impulses as you’ve been doing in Milestone I and in the first half of this milestone. As you will see, it is quite easy to implement: it’s just another force you add to the simulation, without needing to do any separate collision detection or modification of the main simulation loop. On the other hand, the penalty force requires that you choose a value for the spring stiffness \(k\): set it too high and the simulation becomes unstable, unless you decrease the time-step and thereby slow down the simulation. Set it too low and the penalty force is very weak, resulting in a lot of “give” when objects collide, and in the worst case allowing objects to tunnel completely through each other. You also have to choose a value for \(T\). Setting \(T\) to zero means that no force is applied on the objects until they are already interpenetrating, so they will visibly sink into each other during a collision. Setting \(T\) large prevents such artifacts since the springs will have time to repel the objects before they interpenetrate, but at the cost of unnatural-looking action at a distance: objects appear to slow down even though there is still a large gap separating them.

As usual, the penalty force is derived from the potential by taking the gradient with respect to position:

\[
F = -\nabla V = \begin{cases} 
0, & \|\mathbf{n}\| > r_1 + r_2 + T \\
-k(\|\mathbf{n}\| - r_1 - r_2 - T)(\nabla\mathbf{n})^T\hat{\mathbf{n}}, & \|\mathbf{n}\| \leq r_1 + r_2 + T,
\end{cases}
\]
where the precise formula for the rectangular matrix $\nabla n$ depends on the two objects involved (particle–particle, particle–edge, or particle–half-plane; formulas for each case are derived below). Modify PenaltyForce.cpp and implement the gradient of the penalty potential $(\nabla V)^T$ in each of these three cases. You do not need to implement the corresponding Hessians – all test scenes and grading scenes will only use the penalty method with explicit Euler or forward-backward Euler.

### 5.5.3.8 Particle–Particle

For a pair of particles, $n = \mathbf{x}_2 - \mathbf{x}_1$, where $\mathbf{x}_1$ and $\mathbf{x}_2$ are the positions of the first and second particle, respectively, so

$$\nabla n = \begin{pmatrix} -I & I \end{pmatrix}.$$

**Important Note:** This matrix is expressed in local indices, with the left block corresponding to the degrees of freedom for the first particle, and the right block corresponding to those for the second particle. See Theme I, Milestone III for a discussion on local versus global indices.

### 5.5.3.9 Particle–Edge

We have that $n = \mathbf{x}_2 + \alpha (\mathbf{x}_3 - \mathbf{x}_2) - \mathbf{x}_1$, where $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ are the positions of the particle and the edge’s two endpoints. The gradient is a bit tricky to derive, since $\alpha$ depends also on the positions, and since we must take into account that $\alpha$ is clamped to $[0, 1]$. It turns out that the formula (in local coordinates) is

$$\nabla n = \begin{pmatrix} -I & (1 - \alpha)I & \alpha I \end{pmatrix}.$$

### 5.5.3.10 Particle–Half-plane

From Milestone I we have that

$$n = \frac{\mathbf{(x}_h - \mathbf{x}) \cdot \mathbf{n}_h}{\|\mathbf{n}_h\|^2} \mathbf{n}_h,$$

so

$$\nabla n = \frac{-\mathbf{n}_h \mathbf{n}_h^T}{\|\mathbf{n}_h\|^2}.$$
5.6 Collisions: Theme II Milestone III

5.6.1 Introduction

In Theme III, you will learn about a new way of resolving collisions geometrically by treating the colliding particles as if they were a rigid body. The penalty method, geometric method, impulse method, and continuous-time collision detection will then be combined into one robust, hybrid solution to collision detection and response. Test scenes for this milestone are located in a new directory, theme2assets/Milestone03.

5.6.2 New XML Features

This milestone adds the following new feature:

1. The collision node’s functionality has been expanded:

   ```xml
   <collision type="hybrid" maxiters="10" k="100" thickness="0.1"/>
   ```

   The valid values for type are “none”, “simple”, “continuous-time”, “penalty”, and “hybrid”; you will be implementing the hybrid method in this milestone. In addition to the attributes described in Milestones I and II, collision has the new optional attribute maxiters specifying the maximum number of times to try applying impulses before switching to a geometric failsafe, as explained in section 5.6.3.5 below. This attribute has no effect unless type is “hybrid”.

5.6.3 Required Features for Milestone III

5.6.3.1 Handling Multiple Simultaneous Collisions

So far in this theme we have been assuming that doing one pass of pairwise collision detection, followed by collision response in the form of applying impulses to each pair of colliding objects, is enough to produce a collision-free simulation. This assumption does not take into account the possibility of three or more objects colliding simultaneously during a single time-step. Figure 5.11 shows one example where this assumption breaks down: only one collision is detected at first, between particles 1 and 2, but resolving this collision introduces a new one, between
Figure 5.11: Iterative collision response: An example showing that a single round of collision detection and response can be insufficient. Initially, particles 1 and 2 are detected as colliding (left); applying impulses resolves this collision, but introduces a new collision between particles 2 and 3 (middle left). A second iteration of collision detection and response (middle right) is necessary to produce a collision-free simulation at the end of the time-step (right).
particles 2 and 3. Unless we do a second round of collision detection and response, we will fail to stop this second collision, and the time-step will end with the simulation in an interpenetrated state.

One way to handle the problem of simultaneous collisions is to wrap a loop around detection and response, and to keep modifying positions and velocities until there are no collisions detected at the end of the time-step. Here is the algorithm, a modification of that given in Theme II Milestone II, section 4.3:

1. Step the positions \( q^s \) and velocities \( \dot{q}^s \) forward using the numerical integrator to get initial predicted end-of-time-step positions \( q^e_0 \) and velocities \( \dot{q}^e_0 \).

2. Assume the particles move with constant velocity from \( q^s \) to \( q^e_0 \), and perform continuous-time collision detection with \( q(t) = q^s + t\Delta q_0 \), where \( \Delta q_0 = q^e_0 - q^s \).

3. For \( i = 0, 1, \ldots \) until no collisions are detected:

   4. (a) Apply impulses to the predicted velocities \( \dot{q}^e_i \) to get new predicted velocities \( \dot{q}^e_{i+1} = \dot{q}^e_i + M^{-1}I \). The vector \( I \) is the sum of the impulses derived in Milestone I needed to resolve the detected collisions.

   (b) Also modify positions to get new predicted positions: \( q^e_{i+1} = q^e_i + hM^{-1}I \).

   (c) Assume the particles move with constant velocity from \( q^s \) to \( q^e_{i+1} \), and perform continuous-time collision detection with \( q(t) = q^s + t\Delta q_{i+1} \), where \( \Delta q_{i+1} = q^e_{i+1} - q^s \).

If the above loop terminates, then it is guaranteed by construction to result in end-of-time-step position that are collision-free. Unfortunately, it can be shown that for some values of \( COR \), there exist simulations for which infinitely many iterations are necessary to resolve all collisions: every impulse that stops one pair of objects from colliding causes a different pair of objects to collide. Moreover, you might need many, many iterations to fully resolve all collisions in a simulation with a large clump of objects. Even if the loop terminates, it might do so after running for a prohibitively long time. These potential pitfalls motivate us to look for an alternative approach that is guaranteed to handle multiple simultaneous collisions quickly.
5.6.3.2 Geometric Collision Response

Suppose we have some number of particles and edges that we know are about to collide with each other. There’s a very simple thing we can do that is guaranteed to stop all collisions: we can simply freeze all the particles and edges in place. This solution is obviously extremely simplistic; it doesn’t conserve momentum, for instance.

A more sophisticated approach is to treat the set of particles and edges as a **rigid body**: suppose every particle and edge endpoint is connected to every other one by a rigid metal beam. Since distances between particles and edges cannot change, the objects cannot collide. What we need to do, then, is calculate how the rigid body would move after the beams are attached, as a function of how the individual objects moved before attachment. For the purposes of this method, we will assume that the mass of an edge is lumped at its endpoints (so that the span between the endpoints is massless). This assumption allows us to ignore the edge itself in what follows, and work only with particles and edge endpoints (which are also particles).

Suppose we have a collection of particles with start-of-time-step positions \( x_i^s \) and (colliding) end-of-time-step positions \( x_i^e \). We also suppose that the particles move from their old to new positions with constant velocity \( \Delta x_i = x_i^e - x_i^s \). We want to do the following:

1. Treat the particles as if they were part of a rigid body; that is, treat them as if we connected them with rigid beams at the start of the time-step.
2. Step the rigid body forward in time to the end of the time-step.

3. Set each particle’s modified end-of-time-step position \( x_i^m \) to the position dictated by the motion of the rigid body.

4. Also set the particle’s modified end-of-time-step velocity to \( \frac{(x_i^m - x_i^s)}{h} \), where \( h \) is the length of the time-step.

Figure 5.12 shows this algorithm step by step.

If the masses of the particles are \( m_i \), their center of mass is given by

\[
x_{cm}^s = \frac{\sum_i m_i x_i^s}{\sum_i m_i}.
\]

When all masses are equal, the center of mass is just the average of the particle positions. If masses are unequal, the center of mass is a weighted average, with the more massive particles having greater weight. A rigid body’s configuration is completely determined by only three degrees of freedom: the position of the center of mass of the body, and the orientation (rotation) of the body about the center of mass. Similarly, the motion of the rigid body is determined by the velocity \( \Delta x_{cm} \) and angular velocity (speed of rotation) \( \omega_{cm} \) of the center of mass. At the start of the time-step, we want to convert the individual particles into a single rigid body; we do so by calculating \( \Delta x_{cm} \) and \( \omega_{cm} \).

To find \( \Delta x_{cm} \), we invoke conservation of momentum. The momentum of the rigid body has to be equal to the momentum of the individual particles:

\[
\left( \sum_i m_i \right) \Delta x_{cm} = \sum_i m_i \Delta x_i
\]

\[
\Delta x_{cm} = \frac{\sum_i m_i \Delta x_i}{\sum_i m_i}.
\]

Similarly, \( \omega_{cm} \) is derived by looking at conservation of angular momentum. The total angular momentum (about the center of mass) of the individual particles is

\[
L = \sum_i m_i (x_i^s - x_{cm}) \times (\Delta x_i - \Delta x_{cm}),
\]

where \((a,b) \times (c,d)\) is the scalar two-dimensional cross-product \(ad - bc\). We want the angular momentum of the rigid body to have the same value. Angular momentum is related to angular
velocity by the formula

\[ L = I \omega_{cm}, \]

where \( I \) is the moment of inertia \( I = \sum m_i \|x_i^s - x_{cm}\|^2 \) of the rigid body. Therefore

\[ \omega_{cm} = L/I. \]

Now we know how to step the rigid body forward in time: the center of mass translates by \( \Delta x_{cm} \), and the body rotates about the center of mass by \( \omega_{cm} \). To convert the rigid body’s new position into new positions for the individual particles, we use the formula

\[ x_i^e = x_{cm} + \Delta x_{cm} + R_{\omega_{cm}}(x_i^s - x_{cm}) \]

\[ = x_{cm} + \Delta x_{cm} + \cos(\omega_{cm})(x_i^s - x_{cm}) + \sin(\omega_{cm})(x_i^s - x_{cm})^\perp, \]

where \( R_{\omega_{cm}} \) is rotation about the origin by \( \omega_{cm} \), and \((a, b)^\perp = (-b, a)\).

5.6.3.3 Handling Fixed Objects

There is another wrinkle that must be addressed: if the set of colliding objects includes a fixed object (such as a fixed particle, an edge containing an endpoint that’s fixed, or any half-plane), we cannot move the set of objects as a rigid body since the fixed object’s position is not allowed to change. For this milestone, when applying geometric collision response to a set of objects that includes a fixed object, instead of the above set \( x_i^e = x_i^s \) and \( v_i^e = 0 \). There are more clever things you could do (for instance, if the only fixed object is a particle, you might set that particle’s position to be the center of mass, and allow the other particles to rotate about that position) but they are beyond the scope of this milestone.

5.6.3.4 Impact Zones

In the previous section we saw how to take a set of colliding objects and compute end-of-time-step positions for them that are guaranteed to be collision-free, by treating them all as part of one rigid body. What we still need is the big picture: given some start-of-time-step positions \( \mathbf{q}^s \) and end-of-time-step positions \( \mathbf{q}^e \), with some of the objects colliding as they move between these two positions, to which subset of the objects do we apply the geometric response discussed in section 5.6.3.2? Moving all particles and edges in the simulation as a rigid body, even those
Figure 5.13: Impact zones: A number of collisions are detected during a time-step (left). Based on these detected collisions, the particles in the simulation involved in collisions are separated into impact zones (right). Each impact zone is a different color.

not at all involved in a collision, is clearly a poor solution. Treating all objects involved in a collision as part of the same rigid body is also suboptimal; if there are two clumps of collisions in two separate areas of the simulation, we don’t want to couple them unnecessarily by gluing the two clumps together into one rigid body.

Instead, we will split up all particles involved in a collision into one or more impact zones. An impact zone is a set of particles, as well as a boolean flag indicating whether or not a half-plane is colliding with one of the particles in the impact zone. Conceptually, each impact zone is one clump of interconnected collisions, with no particle shared by more than one impact zone. Each impact zone will be treated as a separate rigid body during geometric collision response; Figure 5.13 illustrates the impact zones for an example set of detected collisions.

Here is the concrete algorithm for how to turn a set of detected collisions into disjoint impact zones:

1. First, create an impact zone for each detected collision.

   (a) For each collision detected between particles $i$ and $j$, create an impact zone containing the set of particles $\{i, j\}$. This impact zone does not involve a half-plane.

---

1 We need to keep track of which impact zones involve a half-plane since we must treat such zones specially. See section 5.6.3.3.
Figure 5.14: Iterative geometric collision handling: Some collisions are detected, and impact zones constructed (left). Stepping the simulation forward causes new collisions (middle left), so the impact zones grow (middle). This process repeats a second time (middle right and right).

(b) For each collision detected between particle \( i \) and edge whose endpoints are particles \( j \) and \( k \), create an impact zone containing the set of particles \{\( i, j, k \)\}. This impact zone does not involve a half-plane.

(c) For each collision detected between a particle \( i \) and a half-plane, create an impact zone containing only the lone vertex \{\( i \)\}. This impact zone does involve a half-plane.

2. Two impact zones are not allowed to share the same particle. If any two impact zones \( Z_1 \) and \( Z_2 \) share a particle, they must be *merged*: replace the two zones with a single new zone containing the union of \( Z_1 \) and \( Z_2 \)’s particles. The new zone involves a half-plane if either of the old zones did.

3. Repeat the previous step until all zones are disjoint.

Once we have a set of disjoint impact zones, we can adjust the end-of-time-step positions of the vertices in each zone using geometric collision response. It is important to stress that this response is applied to each zone separately: each zone acts as a different rigid body.

Just as when resolving collisions using impulses, there is a possibility that one iteration of response could cause new collisions: a zone \( Z_1 \) might, while moving as a rigid body, collide into an object not in \( Z_1 \). One possibility discussed in class is to go back to the approach in 5.6.3.1 and apply impulses between the rigid body and the new object. How to apply impulses to rigid bodies will be covered in Theme III; for now, if such a collision is detected, we will *grow* \( Z_1 \) by adding the new object to it, and recompute the end-of-time-step positions for the particles in the new \( Z_1 \). As impact zones grow in this way, it is also possible for them to merge. Here
is the algorithm in more detail; it assumes we have start-of-time-step positions and velocities \( q^s, \dot{q}^s \) and predicted end-of-time-step positions and velocities \( q^e, \dot{q}^e \), and are trying to find collision-free, modified end-of-time-step positions and velocities \( q^m, \dot{q}^m \).

1. Perform continuous-time collision detection using positions \( q^s \) and \( q^e \).

2. Initialize \( q^m = q^e \) and \( \dot{q}^m = \dot{q}^e \).

3. Construct a list of disjoint impact zones \( Z \) from the detected collisions.

4. For each impact zone in \( Z \), apply geometric collision response, using positions \( q^s \) and \( q^m \), and modifying \( q^m \) and \( \dot{q}^m \) for the vertices in those zones.

5. Perform continuous-time collision detection using positions \( q^s \) and \( q^m \).

6. Construct a new list of impact zones \( Z' \) consisting of all impact zones in \( Z \), plus one zone for each detected collision.

7. Merge the zones in \( Z' \) to get disjoint impact zones.

8. If \( Z \) and \( Z' \) are equal, the algorithm is done: \( q^m \) and \( \dot{q}^m \) are the new, collision-free end-of-time-step positions. \( Z \) and \( Z' \) are equal if they contain exactly the same impact zones; impact zones are the same if they contain the same particles and they both involve, or both don’t involve, a half-plane.

9. Set \( Z = Z' \) and go to step 4.

Figure 5.14 illustrates the algorithm. Unlike the interactive impulse algorithm described in section 5.6.3.1, this algorithm is guaranteed to terminate after finitely many iterations: during each iteration, either an impact zone grows, or an impact zone that didn’t involve a half-plane now involves a half-plane. This process must eventually stop (in the worst case, with every single particle contained in one impact zone that involves a half-plane).

5.6.3.5 Putting It All Together: Hybrid Collision Handling

Although geometric collision response is guaranteed to produce collision-free end-of-time-step positions, and conserves momentum and angular momentum, it is important to realize that it
is not physically correct: for instance, a particle that hits a fixed edge at an angle should either
reflect off of the edge (for COR = 1.0), slide along the edge (for COR = 0.0), or reflect at
some angle between these two extremes. In every case, the particle’s motion in the direction
tangential to the edge is unrestricted. When using geometric response, on the other hand, the
particle will come to a dead stop. In general, handling collisions using geometric response alone
results in simulations that look “chunky” and unnatural.

However, geometric response can be a potent tool when combined with iterated impulses.
Instead of iterating applying impulses until no collisions are detected – which could take a
very long time – we cap the loop at a certain fixed maximum number of iterations $n$. After $n$
iterations, if there are still unresolved collisions, we switch to geometric impulses as a failsafe.
Here is the algorithm:

1. Perform collision handling using iterative impulses, as described in 5.6.3.1. Stop after $n$
iterations, or if there are no new detected collisions. The outputs from this process are
new predicted end-of-time-step positions and velocities $q^e_n$ and $\dot{q}^e_n$.

2. If there were still unresolved collisions, perform geometric collision handling, using these
new predicted quantities. The outputs from this step are guaranteed collision-free posi-
tions and velocities $q^m$ and $\dot{q}^m$.

If objects in a simulation become clumped in very close proximity, many simultaneous collisions
occur and it becomes likely that the failsafe (step 2) is necessary to resolve these collisions. To
try to create some breathing room between nearby objects and prevent this situation from
occurring too often, we will add a final ingredient to our hybrid method: a weak penalty force
that repels objects that are near-touching. You do not need to implement this penalty force:
the starter code automatically adds it to simulations with hybrid collision detection.

Edit HybridCollisionHandler.cpp and implement steps 1 and 2. To aid with debugging, the
oracle will check several of your intermediate results and report (on the console) any discrep-
ancies:

- The predicted end-of-time-step positions and velocities after iterative impulses (step 1),
- The set of impact zones during each iteration of geometric collision handling,
• The predicted end-of-time-step positions and velocities after each iteration of geometric collision handling.

5.6.3.6 Further Reading

The hybrid method presented above is adapted from algorithms described in the following papers for handling collisions between cloth and objects in 3D simulations. Over the course of this theme, you have learned many of the main ideas that make up these algorithms, and are now well equipped to further explore the details:


5.7 Collisions: Theme II Milestone IV

5.7.1 Introduction

In Milestone IV, you will design your own algorithm for accelerating collision detection in scenes containing a large number of particles or edges. Test scenes for this milestone are located in a new directory, theme2assets/Milestone04.

5.7.2 New XML Features

This milestone adds the following new feature:

1. The new collisiondetection node specifies how collision detection is performed during the simulation:

   <collisiondetection type="allpairs"/>
The valid values for type are “allpairs” (the default) and “contest”. allpairs is already implemented in the starter code, and performs a brute-force check of all pairs of objects to see if any two of them are colliding. You will be implementing the contest algorithm in this milestone.

**5.7.3 Required Features for Milestone IV**

**5.7.3.1 Collision Detection Contest**

Edit ContestDetector.cpp and implement any algorithm you like for speeding up collision detection when handling collisions using the penalty method with thickness 0. In other words, given start-of-timestep positions, you are to return all particle–particle, particle–edge, and particle–halfplane pairs that might be overlapping at those positions. You do not need to perform continuous-time collision detection, and you do not need to check if particles are approaching, since the penalty method only cares about overlap at the start of the time-step. The penalty force will then take your list of potentially overlapping pairs, compute a force for that pair, and apply it to the system. Note that the computed force will be zero if the pair is not actually overlapping.

One solution is to simply return all pairs of objects in the scene as potentially overlapping. This is the solution that is implemented as “allpairs” collision detection in the starter code. It is very slow – $O(n^2)$ – since a gradient has to be computed for all of these pairs, even those that aren’t overlapping.

Another solution is to compute the distance between all pairs of objects, and only return those that are actually overlapping. No time is wasted computing unnecessary gradients, but the detection itself is now very slow – again $O(n^2)$. You want to design an algorithm in between these two extremes – one that can quickly determine all pairs that are likely to be overlapping, without including too many false positives.

We will run your algorithm on a gauntlet of three very large test scenes, and measure the total time it takes your code to simulate the scenes. We will compare your algorithm’s time against that of the other submissions, and your milestone grade will be determined by how well your algorithm performs. Here are the details of the competition.
• You may not sacrifice correctness for speed; in other words, your collision detection must be *conservative*. If two objects actually overlap during a time-step, you *must* include that pair in your list of detected pairs. If they do not overlap, you *may* include that pair in the list. The oracle can be used to check that your code does not miss collisions.

• You may not change any files in the starter code except `ContestDetector.cpp` and `ContestDetector.h` (you may also create new source files containing helper code, if you’d like). We will overwrite all other files with those supplied in the starter code before compiling your contest entry. There are many inefficiencies in the starter code that could be optimized – but the focus of this milestone is solely on collision detection.

• You are encouraged to use the internet, class notes, research papers, etc. to find potentially useful ideas or algorithms. However, all non-starter code you submit must be entirely your own. You may not link to any external libraries except for those already used by the starter code (e.g. Eigen).

• The three test scenes will include
  
  – A scene very similar to the box-of-balls scene included with the starter code,
  
  – A scene very similar to the ribbon-pile scene included with the starter code,
  
  – A scene very similar to the best submitted creative scene.

You should strive to design a well-rounded algorithm that performs well for both provided scenes.

• For each of the three test scenes, we will run your code and record the time taken. If any of the following occurs, we will use the time taken by the oracle’s implementation of “allpairs” instead:

  – your code fails the oracle due to unacceptably large residual (ie, you miss a collision),
  
  – your code fails to compile,
  
  – your code takes longer than the time taken by the oracle’s implementation of “allpairs”,

your code crashes or fails to complete the simulation for any other reason.

Each scene is worth 28% of your milestone grade. For each scene, your grade is determined as follows: if $T$ is your total time, $A$ is the total time of the oracle running “allpairs”, $M$ is the fastest total time in the class, $r$ the number of submissions slower than yours, and $N$ the total number of submissions, your grade is given by the formula

$$14 \left( \frac{A - T}{A - M} \right)^{1/3} + 14 \frac{r}{N - 1}.$$

### 5.8 Rigid Body Simulation: Theme III Milestone I

#### 5.8.1 Introduction

In Theme III we will explore methods for simulating rigid bodies – objects in which the distance between any two points is fixed for all time. Rigid bodies have a number of interesting applications ranging from their use in robot motion planning to studying the time evolution of tops. Rigid bodies are also one of the most common simulatable primitives in video game physics engines.

Rigid bodies see heavy use as they possess many favorable numerical properties; simulating similar systems with masses and springs or finite elements will lead to very stiff systems and hence strict time-step limits. As a reduced coordinate representation, rigid bodies also have the potential to yield large savings due to the reduction in the number of degrees of freedom.

The use of reduced or generalized coordinates can be viewed as a constraint enforcement method. For example, one can encode the position of a pendulum with one variable that changes with time - the deflection of the pendulum arm from the vertical. The position of the pendulum bob can be recovered using simple trigonometry, the angle of deflection, and the length of the arm. By construction, the inextensibility of the arm cannot be violated. In contrast, if one were to use Cartesian coordinates, the inextensibility of the constraint would have to be enforced in some manner. Usually reduced coordinates are difficult to write down for complex constrained systems, but for rigid bodies they prove both manageable and advantageous.

In this first milestone we will explore the dynamics of rigid bodies in the absence of collisions.
Figure 5.15: The hull of a rigid body: We describe the shape of a rigid body by a sequence of vertices. Note the clockwise specification of vertices.

### 5.8.2 New XML Features

This milestone introduces the following new simulation commands:

1. The `simtype` node specifies the type of this simulation:

   ```xml
   <simtype type="rigid-body"/>
   ```

   The valid values for `type` are “particle-system” and “rigid-body”. Setting the type to “particle-system” indicates that this scene file describes a particle simulation as implemented in the previous two themes. In this theme, we will use a setting of “rigid-body”.

2. The `rigidbodyintegrator` node specifies both the time integration technique and the time-step to use for this rigid body simulation:

   ```xml
   <rigidbodyintegrator type="explicit-euler" dt="0.001"/>
   ```

   The `type` attribute specifies which time integration technique to use: valid values are “explicit-euler” and “forward-backward-euler”. The `dt` attribute specifies the time-step to use. `dt` expects a positive scalar.

3. The `rigidbodyvertex` node creates a new vertex that the user can assign to a rigid body:

   ```xml
   <rigidbodyvertex x="-5" y="-1" m="1"/>
   ```

   The `x` and `y` attributes specify the position of the vertex. Both `x` and `y` expect scalar values. The `m` attribute specifies the mass of the vertex and must be a positive scalar.
4. The `rigidbody` node creates a new rigid body:

   `<rigidbody p="0" p="1" p="2" p="3" vx="0" vy="0" omega="0" r="0.1"/>`

   Each `p` attribute references a `rigidbodyvertex` and adds a new vertex to this `rigidbody`'s hull. The vertices that define the (generally non-convex) hull of this `rigidbody` are specified in clockwise order. The first and final vertex are also connected. See Figure [5.15](#). The center of mass, the total mass, and the moment of inertia of the rigid body are computed from the positions and masses of these vertices. The `vx` and `vy` attributes specify the initial velocity of the center of mass of this `rigidbody`. The `omega` attribute specifies the initial angular velocity about the center of mass of this `rigidbody`. The `r` attribute specifies the radius of this `rigidbody` for rendering and collision-detection purposes.

5. The `rigidbodyspringforce` node creates a spring force that acts on rigid bodies:

   `<rigidbodyspringforce i="0" pix="1" piy="1" j="1" pjx="-1" pjy="1" k="1" l0="4"/>`

   The `i` and `j` nodes specify which rigid bodies this force acts between. If either value is set to -1, this indicates that the corresponding endpoint is fixed in space and not attached to a rigid body. If an endpoint is attached to a rigid body, the `pix`, `piy`, `pjx`, and `pjy` nodes specify where on each rigid body the spring is attached relative to the center of mass (assuming the original orientation). If instead the corresponding endpoint is specified as fixed, these values specify the position in space where the spring endpoint is fixed to. The `k` node specifies the stiffness of the spring and the `l0` node specifies the rest-length of the spring.

6. The `rigidbodygravityforce` node creates a (constant near-earth) gravity force that acts on all rigid bodies:

   `<rigidbodygravityforce fx="0.0" fy="-9.81"/>`

   The `fx` and `fy` attributes specify the `x` and `y` components of the force, respectively.

7. The `rigidbodywindforce` node creates a wind-like force that acts on all rigid bodies:
The positive scalar \( \beta \) attribute scales the overall strength of the force. The \( fx \) and \( fy \) attributes specify the \( x \) and \( y \) components of the force, respectively. The integer \( \text{pointsperedge} \) attribute specifies the number of quadrature points per edge to use when computing this force (see description below).

This milestone introduces the following new rendering commands:

1. The \textit{rigidbodycolor} node sets the color of an entire rigid body:

\[
<\text{rigidbodycolor} \ \text{body}="1" \ \text{r}="0.0" \ \text{g}="0.4" \ \text{b}="0.0"/>\]

The \texttt{body} attribute specifies which rigid body this color applies to. The \texttt{r}, \texttt{g}, and \texttt{b} nodes specify the color of the body. Their values must be between 0 and 1.

2. The \textit{rigidbodyspringcolor} node sets the color of a spring:

\[
<\text{rigidbodyspringcolor} \ \text{spring}="4" \ \text{r}="0.2" \ \text{g}="0.0" \ \text{b}="0.8"/>\]

The \texttt{spring} attribute specifies which spring this color applies to. The \texttt{r}, \texttt{g}, and \texttt{b} nodes specify the color of the spring. Their values must be between 0 and 1.

5.8.3 Rigid Body Dynamics

A rigid body is a collection of masses (or a continuum, but here we consider only a finite number of points) in which the distance between any two masses remains fixed for all time. Intuitively, the two transformations that preserve this invariant are translations and rotations. Therefore, not surprisingly, we will find that we can decompose the dynamics of a rigid body into two components: a component associated with the center of mass that behaves like a point mass, and a component associated with rotations about the center of mass.

These notes closely follow the presentations in [Baraff, 2001], [Goldstein \textit{et al.}, 2002], and [Taylor, 2005]. Please see these references for additional details.
5.8.3.1 Body Space, World Space, and Center of Mass

Consider a 2D rigid body composed of $N$ points masses lying in a 2D plane. Recall that we define the center of mass of a collection of masses as

$$
\mathbf{x} = \frac{\sum_i m_i \mathbf{r}_i}{\sum_i m_i} = \frac{\sum_i m_i \mathbf{r}_i}{M}
$$

where $m_i$ is the mass of the $i^{th}$ particle, $\mathbf{r}_i$ is the position in world space of the $i^{th}$ particle, and $M = \sum_i m_i$ is the total mass.

Define body space as a 2D orthonormal coordinate system with the origin placed at the body’s center of mass and in which the body is considered to have no rotation (see Figure 5.16). If $\mathbf{r}_i^0$ is the position of any point on the rigid body in body space, then the point’s position in world space is given by

$$
\mathbf{r}_i = \mathbf{R}(t)\mathbf{r}_i^0 + \mathbf{x}(t)
$$

where $\mathbf{R}(t)$ is a $2 \times 2$ rotation matrix describing the orientation of the rigid body at time $t$, and $\mathbf{x}(t)$ is the position of the body’s center of mass at time $t$. As $\mathbf{R}$ is purely a rotation, we can simply store a single angle $\theta$ to reconstruct $\mathbf{R}$.

What does this construction gain us? At any time $t$, if we know the orientation of the body $\theta$ and the center of mass’ position $\mathbf{x}$, we can compute the position of any particle on the rigid body. Instead of simulating $2N$ degrees of freedom, we need only simulate 3. Furthermore, as our representation only admits rigid motions, that is rotations and translations, we do not need to enforce the rigidity constraint with soft constraints (e.g. stiff forces), hard constraints (e.g. Lagrange multipliers), or some other technique.
5.8.3.2 Linear and Angular Velocity

We can compute the velocity of a particle attached to a rigid body by taking the time derivative of (5.6):

$$\dot{r}_i = \dot{R}(t)r_i^0 + \dot{x}(t)$$

Note that $r_i^0$ is fixed throughout the simulation and thus its time derivative is 0. $\dot{x}$ is simply the velocity of the rigid body’s center off mass, but how do we compute $\dot{R}$? Before proceeding, let us examine the columns of $R$.

Component-wise, we can write $R$ as:

$$R = \begin{pmatrix} R_{xx} & R_{yx} \\ R_{xy} & R_{yy} \end{pmatrix}$$

Consider the action of $R$ on the Cartesian $x$-axis:

$$R\dot{x} = \begin{pmatrix} R_{xx} & R_{yx} \\ R_{xy} & R_{yy} \end{pmatrix} \begin{pmatrix} 1 \\ 0 \end{pmatrix} = \begin{pmatrix} R_{xx} \\ R_{xy} \end{pmatrix}$$

That is, the first column of $R$ is simply the body space $x$-axis rotated into world space! Similarly, we find that the second column of $R$ is the body space $y$-axis rotated into world space. Therefore,
if we can compute how these vectors evolve instantaneously, we can compute $\dot{\mathbf{R}}$. In order to compute these quantities, we will first derive a more general 3D result.

Consider a rigid body rotating about its center of mass and some axis $\omega$ with angular velocity $\omega = |\omega|$ (see Figure 5.17). Consider a vector pointing from the center of mass to some point on the body, say $\mathbf{r}'_i = \mathbf{Rr}_0 = (\mathbf{r}_i - \mathbf{x}) = \mathbf{p}_\text{perp} + \mathbf{p}_\text{para}$, where $\mathbf{p}_\text{perp}$ is perpendicular to $\omega$ and $\mathbf{p}_\text{para}$ is parallel to $\omega$. $\mathbf{p}_\text{para}$ is unaffected by the rotation, while $\mathbf{p}_\text{perp}$ will trace out a circle. Simple trigonometry gives that the radius of the circle is $|\mathbf{r}_i - \mathbf{x}| \sin \phi$, where $\phi$ is the angle between $(\mathbf{r}_i - \mathbf{x})$ and $\omega$. Thus, the instantaneous speed of any point on this circle is given by $|\omega| |\mathbf{r}_i - \mathbf{x}| \sin \phi$. Furthermore, the direction of the velocity of this point is perpendicular to both $\omega$ and $(\mathbf{r}_i - \mathbf{x})$. What operation gives this magnitude and this direction? The cross product! Therefore, the velocity of $\mathbf{r}_i - \mathbf{x}$ is given by $\omega \times (\mathbf{r}_i - \mathbf{x})$.

As a notational convenience, note that we can represent a cross product as a matrix-vector multiplication:

$$\mathbf{a} \times \mathbf{b} = \begin{pmatrix} a_y b_z - a_z b_y \\ a_z b_x - a_x b_z \\ a_x b_y - a_y b_x \end{pmatrix} = \begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix} \begin{pmatrix} b_x \\ b_y \\ b_z \end{pmatrix} = \mathbf{a}^* \mathbf{b}$$

Returning to the problem of computing $\dot{\mathbf{R}}$, first note that translations of the rigid body do not change $\mathbf{R}$, and thus we need only consider changes to $\mathbf{R}$ due to the angular velocity. Invoking the above result:

$$\dot{\mathbf{R}} = \left( \omega \times \begin{pmatrix} R_{xx} \\ R_{xy} \end{pmatrix} \right) \omega \times \begin{pmatrix} R_{yx} \\ R_{yy} \end{pmatrix} = \omega^* \mathbf{R}$$

Thus, the world space velocity of $i^{th}$ particle in the rigid body is $\dot{\mathbf{r}}_i = \omega(t)^* \mathbf{R}(t) \mathbf{r}_i^0 + \dot{\mathbf{x}}(t)$. That is, if in addition to $\mathbf{x}(t)$ and $\theta(t)$ we know $\dot{\mathbf{x}}(t)$ and $\omega(t)$, we can recover the velocity of any point on the rigid body! Therefore, we can represent the entire state of the rigid body with these four quantities.

If we wanted to simulate rigid bodies without any forces, the above would be enough; $\dot{\mathbf{x}}$ and $\omega$ would remain constant throughout the simulation, and we would just update $\mathbf{x}(t)$ and $\theta(t)$ using these constant values. This situation is not terribly interesting, however, so we will now see how Newton’s third law ($\mathbf{F} = m\mathbf{a}$) looks when phrased in terms of our reduced coordinates.
5.8.3.3 Forces and Torques

5.8.3.4 Center of Mass Acceleration

To determine how a force effects the center of mass’ velocity, $\dot{x}$, we will start by computing the total momentum of the rigid body. The total momentum $P$ of a rigid body is given by:

$$P = \sum_i m_i \dot{r}_i = \left( \sum_i m_i \right) \frac{\sum_i m_i \dot{r}_i}{\sum_i m_i} = M \dot{x}$$

That is, the momentum of the system can be computed by viewing all of the body’s mass as moving with the center of mass. Taking the derivative of this quantity gives us the center of mass’ acceleration:

$$M \ddot{x} = \sum_i m_i \ddot{r}_i = \sum_i f_i$$

$f_i$ is the total force acting on mass $i$. We have to be a little careful going forward, as this force includes both external forces (e.g. gravity) as well as the internal forces acting within the rigid body. Let us assume that each point $j$ exerts a force on each other point $i$ to maintain the rigidity constraint. That is, we can write the force on particle $i$ as $f_i = f_i^{ext} + \sum_{j \neq i} f_{ij}$ where $f_i^{ext}$ is the sum of the external forces acting on $i$, and $f_{ij}$ is the internal force on $i$ from point $j$.

Substituting this sum into the rate of change of momentum:

$$M \ddot{x} = \sum_i f_i = \sum_i (f_i^{ext} + \sum_{j \neq i} f_{ij}) = \sum_i f_i^{ext} + \sum_i \sum_{j \neq i} f_{ij}$$

Let us examine $\sum_i \sum_{j \neq i} f_{ij}$ in more detail. We can rewrite this sum and invoke Newton’s third law (‘for every action there is an opposite and equal reaction’), giving:

$$\sum_i \sum_{j \neq i} f_{ij} = \sum_i \sum_{j > i} (f_{ij} + f_{ji}) = \sum_i \sum_{j > i} (f_{ij} - f_{ij}) = 0$$

That is, provided the internal constraint forces obey Newton’s third law, to compute the acceleration of the center of mass, we simply sum the external forces acting on each particle of the body:

$$M \ddot{x} = \sum_i f_i^{ext} = F$$
5.8.3.5 Angular Acceleration

To determine the angular acceleration of a rigid body, we will first compute the time derivative of the angular momentum measured with respect to the origin. The total angular momentum of rigid body is given by

\[ L = \sum_i r_i \times p_i \]

and thus we compute the time derivative as:

\[ \dot{L} = \sum_i \dot{r}_i \times p_i + \sum_i r_i \times \dot{p}_i \]

In the first term, the velocity of a particle is parallel to its momentum, so this term is 0. In the second term, \( \dot{p}_i = f_i = f_i^{\text{ext}} + \sum_{j \neq i} f_{ij} \) is the total force acting on the point, which as in the previous section includes both the external forces on the point and the internal constraint-maintaining forces. Making this substitution, we find:

\[ \dot{L} = \sum_i r_i \times f_i^{\text{ext}} + \sum_i r_i \times \sum_{j \neq i} f_{ij} \]

Let us examine the second term in more detail. We can rearrange the sum and invoke Newton’s third law to obtain:

\[ \sum_i r_i \times \sum_{j \neq i} f_{ij} = \sum_i \sum_{j \neq i} r_i \times f_{ij} = \sum_{j > i} \sum_i (r_i \times f_{ij} + r_j \times f_{ji}) = \sum_{j > i} \sum_i ((r_i - r_j) \times f_{ij}) \]

When is this sum always 0? When \( r_i - r_j \) is parallel to \( f_{ij} \). That is, when all constraint forces are central and obey Newton’s third law, the time rate of change in angular momentum is simply the sum of all external torques:

\[ \dot{L} = \sum_i r_i \times f_i^{\text{ext}} \]

We can further decompose the angular momentum into a component associated with the center of mass’ motion and a component associated with motion (spin) about the center of mass. Let \( r_i' = R(t) r_i^0 \) denote the body space coordinate of particle \( i \) rotated into world space by the body’s orientation. Substituting a particle’s position expressed relative to the center of
mass into the formula for angular momentum:

\[ L = \sum_i r_i \times p_i = \sum_i (x + r'_i) \times m_i (\dot{x} + \dot{r}'_i) \]

\[ = \sum_i x \times m_i \dot{x} + \sum_i x \times m_i r'_i + \sum_i r'_i \times m_i \dot{x} + \sum_i r'_i \times m_i r'_i \]

\[ = x \times \left( \sum m_i \dot{x} \right) + x \times \left( \sum m_i r'_i \right) + \left( \sum m_i r'_i \right) \times \dot{x} + \sum m_i r'_i \times m_i r'_i \]

\[ = x \times \mathbf{P} + x \times \left( \sum m_i r'_i \right) + \left( \sum m_i r'_i \right) \times \dot{x} + \sum m_i r'_i \times \mathbf{p}'_i \]

Let us look more closely at the two terms in parenthesis. Expanding \( \sum_i m_i r'_i \) gives:

\[ \sum_i m_i r'_i = \sum_i m_i (r_i - \mathbf{x}) = \]

\[ \sum_i m_i r_i - \sum_i m_i \mathbf{x} = \left( \sum m_i \right) \sum_i m_i r_i - \left( \sum m_i \right) \mathbf{x} = M \mathbf{x} - M \mathbf{x} = 0 \]

\( \sum_i m_i r'_i \) is simply the time derivative of this function, and hence is also 0. Thus, the total angular momentum of a rigid body with respect to the origin is given by:

\[ L = x \times \mathbf{P} + \sum_i r'_i \times m_i \mathbf{p}'_i = \mathbf{L}^{\text{point}} + \mathbf{L}^{\text{spin}} \]

Conveniently, the angular momentum decomposes into a component associated with the center of mass and the total momentum of the system, and a component associated with velocity relative to the center of mass. We will now compute \( \dot{\mathbf{L}}^{\text{point}} \), and use this to obtain a simple formula for \( \dot{\mathbf{L}}^{\text{spin}} \).

\[ \dot{\mathbf{L}}^{\text{point}} = \dot{x} \times \mathbf{P} + x \times \dot{\mathbf{P}} = x \times \mathbf{F} \]

Here we have used that \( \dot{x} \) is parallel to \( \mathbf{P} \) and that \( \dot{\mathbf{P}} = \mathbf{F} \) as proved in the previous section.

We now compute \( \dot{\mathbf{L}}^{\text{spin}} \) as

\[ \dot{\mathbf{L}}^{\text{spin}} = \dot{\mathbf{L}} - \dot{\mathbf{L}}^{\text{point}} = \sum_i r_i \times \mathbf{f}^{\text{ext}}_i - x \times \mathbf{F} \]

\[ = \sum_i (x + r'_i) \times \mathbf{f}^{\text{ext}}_i - x \times \mathbf{F} = x \times \sum_i \mathbf{f}^{\text{ext}}_i + \sum_i r'_i \times \mathbf{f}^{\text{ext}}_i - x \times \mathbf{F} \]

\[ = x \times \mathbf{F} + \sum_i r'_i \times \mathbf{f}^{\text{ext}}_i - x \times \mathbf{F} = \sum_i r'_i \times \mathbf{f}^{\text{ext}}_i \]
This is a remarkable result! If we compute the angular momentum in the non-inertial (accelerating) reference frame of the center of mass, the rate of change of the angular momentum takes the simple form \( \dot{L}_{\text{spin}} = \sum_i r_i' \times f_{i}^{\text{ext}} \).

We now restrict ourselves to 2D, and without loss of generality consider a body undergoing only a rotational motion with instantaneous angular velocity \( \omega \) (as we know how to compute torques in the non-inertial frame of the center of mass). We can compute the velocity of the \( i^{th} \) point as:

\[
\dot{r}_i' = \begin{bmatrix}
\dot{x} & \dot{y} & \dot{z}
\end{bmatrix} = \begin{bmatrix}
0 & 0 & \omega
\end{bmatrix} \begin{bmatrix}
-x_i \\
x_i \\
0
\end{bmatrix}
\]

The angular momentum of this particular point is given by (in 2D a scalar) \( m_i r_i' \times \dot{r}_i' \). Expanding this cross product we find that:

\[
m_i r_i' \times \dot{r}_i' = m_i \begin{bmatrix}
\dot{x} & \dot{y} & \dot{z}
\end{bmatrix} = m_i \begin{bmatrix}
0 & 0 & \omega
\end{bmatrix} \begin{bmatrix}
-x_i \\
x_i \\
0
\end{bmatrix} = m_i r_i' \cdot \dot{r}_i' \omega
\]

We can thus express the total angular momentum in the center of mass frame as

\[
L_{\text{spin}} = \sum_i m_i r_i' \times \dot{r}_i' = \left( \sum_i m_i r_i' \cdot \dot{r}_i' \right) \omega = I \omega
\]

where \( I = \sum_i m_i r_i' \cdot r_i' \) is called the moment of inertia. In 2D, this value is a constant throughout the simulation. To see that \( I \) is constant, rotate the entire body about the center of mass:

\[
I = \sum_i m_i (r_i' \circ R)^T R r_i' = \sum_i m_i (r_i')^T R^T R r_i' = \sum_i m_i (r_i')^T r_i' = \sum_i m_i (r_i')^T R r_i' = \sum_i m_i (r_i')^T r_i'
\]

as a rotation is an orthogonal matrix (\( R^T = R^{-1} \), and thus \( R^T R = Id \)).

In conclusion, we find that \( \dot{L}_{\text{spin}} = I \dot{\omega} = \sum_i r_i' \times f_{i}^{\text{ext}} = \Gamma \), which gives us a simple expression for \( \dot{\omega} \). Note that here we rely on the fact that \( I \) is constant in time (a fact not true in 3D).

### 5.8.3.6 Equations of Motion

To summarize, the equations of motion for our reduced coordinate representation of a rigid body are given by:
\[
\frac{d}{dt} \begin{pmatrix}
x \\
\dot{x} \\
\theta \\
\omega
\end{pmatrix} = \begin{pmatrix}
\dot{x} \\
\omega \\
F/M \\
\Gamma/I
\end{pmatrix}
\]

In two dimensions, \( x, \dot{x}, \) and \( F \) are vectors of length 2. \( \theta, \omega, \Gamma, M, \) and \( I \) are scalars. From here, one can utilize a preferred time integration technique.

5.8.4 Required Features for Theme III

5.8.4.1 Computation of Total Mass, Center of Mass, Moment of Inertia

Please edit `RigidBodies/RigidBody.cpp` and complete the `computeTotalMass`, `computeCenterOfMass`, and `computeMomentOfInertia` methods. The interface for these methods is documented in the header file. The oracle will check and grade these values for all scenes. If your values are incorrect, the oracle will print the correct values.

5.8.4.2 Computation of Momentum

Please edit `RigidBodies/RigidBody.cpp` and complete the `computeTotalMomentum` method. The interface for this method is documented in the header file. The oracle will check and grade this value for all scenes. If your value is incorrect, the oracle will print the correct value.

5.8.4.3 Computation of Angular Momentum

In Section 5.8.3.5 we decomposed the total angular momentum of a rigid body into \( L = L_{\text{point}} + L_{\text{spin}} \). Please edit `RigidBodies/RigidBody.cpp` and complete the `computeCenterOfMassAngularMomentum` and `computeSpinAngularMomentum` methods. The interface for these methods is documented in the header file. The oracle will check and grade these values for all scenes. If your values are incorrect, the oracle will print the correct values.

5.8.4.4 Computation of Kinetic Energy Components

In Section 5.8.3.5 we separated the angular momentum into the components \( L = L_{\text{point}} + L_{\text{spin}} \). Please derive a similar decomposition for the kinetic energy of a rigid body; that
is, separate the kinetic energy into components $T = T^{\text{point}} + T^{\text{spin}}$. Please implement these functions in the `computeCenterOfMassKineticEnergy` and `computeSpinKineticEnergy` functions of `RigidBody.cpp`. These values are serialized and graded. If your values are incorrect, the oracle will print the correct values.

### 5.8.4.5 Explicit and Forward-Backward Euler

Please edit the `stepScene` methods of `RigidBodies/RigidBodyExplicitEuler.cpp` and `RigidBodies/RigidBodyForwardBackwardEuler.cpp` to implement explicit Euler and forward-backward Euler respectively. The documentation for these methods is included in the source files. After implementing these methods, you should be able to pass all tests. As in the first theme, your forward-backward Euler implementation should be implicit in velocity during the position update.

### 5.8.4.6 Near-Earth Gravity Force

Please implement a force corresponding to the (pointwise) potential $U(r_i) = -m_i g \cdot r_i$. As this force acts on all points of the rigid body, to simplify computations, work out the total potential energy, force, and torque on the rigid body due to this force:

$$U^{\text{body}} = \sum_i U(r_i)$$

$$F^{\text{body}} = \sum_i -\nabla U(r_i)$$

$$\Gamma^{\text{body}} = \sum_i r_i' \times -\nabla U(r_i)$$

The resulting formulas are very simple and intuitive.

Please implement `computePotentialEnergy` and `computeForceAndTorque` in `RigidBodies/RigidBodyGravityForce.cpp` (note that the potential energy is serialized and graded). After implementing these methods, all tests under `theme3assets/gravitytests/` should pass.

### 5.8.4.7 Spring Force

Please implement a force corresponding to the potential $U(r_i, r_j) = \frac{1}{2} k (|r_j - r_i| - l_0)^2$, where $r_i$ and $r_j$ are either points on a rigid body or fixed points in space. If the endpoint is a rigid
body, you will have to compute both the force and the corresponding torque. Please fill in the implementations of `computePotentialEnergy` and `computeForceAndTorque` in `RigidBodies/RigidBodySpringForce.cpp` (note that the potential energy is serialized and graded). The interfaces for these functions are documented in the corresponding header. After implementing these methods, you should be able to pass all tests under `theme3assets/springtests/`.

Note: Please be careful with the special case where the spring length is 0 and the rest length $l_0$ is 0 (can you see where in the force computation this will cause trouble?). You will have to explicitly check for this case and exert no force when it is detected.

### 5.8.4.8 Wind Force

We will implement a force that approximates the action of ‘wind’ on a rigid body. The magnitude of this force should scale with the difference between the wind’s velocity and the velocity of some point on the edge in the normal direction, that is with $(v_{\text{wind}} - v_j) \cdot n_i$. The velocity along an edge of a rigid body is not constant, so we will have to subdivide the edge into at least two regions when computing the action of this force. See Figure [5.18](#). To summarize, given a rigid body with $N$ edges and $P$ sample points per edge:

$$F_{\text{wind}} = \sum_{i=0}^{N-1} \sum_{j=0}^{P-1} \beta \frac{l_i}{P} (v_{\text{wind}} - v_j) \cdot n_i$$

$\beta$ scales the magnitude of the force, and $l_i/P$ is the fraction of length assigned to each sample point of the $i^{th}$ edge. As the vertices of the rigid body are specified in clockwise order, one can easily compute the outward normal $n_i$ for a given edge.

When implementing this force, you will have to compute the torque at each sample point.
on each edge. Please implement this force in RigidBodies/RigidBodyWindForce.cpp.

5.8.5 Food For Thought

Answers to these questions are not required, we have included them only for your personal edification.

1. Did we have to choose the center of mass to encode the body’s translation? What would happen if we selected a different point in body space?

2. Can you identify where in this derivation we used the assumption of rigidity?

5.8.6 Complications in moving to 3D

The complexity of a rigid body simulation jumps significantly from 2D to 3D, be warned! Complications include:

1. The axis of rotation is now a three vector.

2. The moment of inertia is dependent on orientation (it is a tensor).

3. The derivative of angular momentum has extra terms because the inertia tensor changes over time.

4. The representation of orientation is no longer a single angle. There are many options, all with tradeoffs:

   (a) Euler angles: Simple (only three scalars), but suffers from ‘gimbal lock.’ There are also 24 possible conventions for representing orientation with Euler angles, and this ambiguity can be confusing.

   (b) Rotation matrix: Simple, but contains six redundant degrees of freedom and can drift from a rotation.

   (c) Quaternion: Requires only four scalars, easy to ‘renormalize’ to correct for drift. Renormalization is not free, however. Conceptually more difficult to work with.
5.9 Rigid Body Simulation: Theme III Milestone II

5.9.1 Introduction

In Theme III Milestone I we derived the equations of motion for a rigid body and explored the simulation of rigid body dynamics in the absence of collisions. In Theme II we explored various methods for resolving collisions, and encountered many shortcomings with these methods (recall the problems associated with the rigidification failsafe). In this milestone we will bring together ideas from these previous milestones and derive a method for rigid body contact. In particular, the response methods in this milestone can gracefully handle breaking contact in the face of multiple co-dependent collisions.

The methods in this milestone depend on more advanced numeric techniques than previous assignments; we will make use of both a linear complimentariry problem (LCP) solver, as well as a convex quadratic-program solver.

5.9.2 New XML Features

This milestone introduces the following new simulation features:

1. The `rigidbodycollisionhandling` feature specifies the detection and response methods to use:

   `<rigidbodycollisionhandling detection="all-pairs" response="lcp"/>`

   The `detection` attribute specifies which collision detection method to use; the only valid value for this assignment is `all-pairs`. The `response` method specifies which response method to use; the only valid values for this assignment are `lcp` and `velocity-projection`.

2. We have added a `fixed` attribute to the `rigidbody` feature:

   `<rigidbody p="40" p="41" p="42" p="43" vx="0" vy="0" omega="0"
   r="0.5" fixed="1"/>`

   If `fixed` is set to 1, forces and contact impulses will not effect the body.
5.9.3 The Linear Complementarity Problem

5.9.3.1 Problem Formulation

Consider a system of rigid bodies in contact at \( k \) points. In order to prevent interpenetration, we would like to compute a set of \( k \) impulses applied at these contact points that yield separating relative velocities. That is, for each contact \( i \), we seek impulses that yield  
\[
\dot{\mathbf{g}}_i = \mathbf{n}_i \cdot \left( \dot{\mathbf{r}}_i^+ + \dot{\mathbf{r}}_q^+ - \dot{\mathbf{r}}_i^- - \dot{\mathbf{r}}_q^- \right) \geq 0.
\]

We denote the relative velocity at the \( i^{th} \) contact point by \( \dot{\mathbf{g}}_i \), the contact normal at the \( i^{th} \) contact by \( \mathbf{n}_i \), and the post-impulse world-space velocities of rigid bodies \( p \) and \( q \) at the \( i^{th} \) contact by \( \dot{\mathbf{r}}_i^+ \) and \( \dot{\mathbf{r}}_q^+ \) (a \( - \) superscript denotes pre-impulse). Denote the change in velocity due to the action of all impulses at the \( i^{th} \) contact point on each body by \( \delta \dot{\mathbf{r}}_i^\text{ip} \) and \( \delta \dot{\mathbf{r}}_{iq}^\text{ip} \). With this notation, the \( i^{th} \) constraint becomes  
\[
\dot{\mathbf{g}}_i = \mathbf{n}_i \cdot \left( \dot{\mathbf{r}}_i^- + \dot{\mathbf{r}}_q^- \right) + \mathbf{n}_i \cdot \left( \delta \dot{\mathbf{r}}_i^\text{ip} - \delta \dot{\mathbf{r}}_{iq}^\text{ip} \right) \geq 0
\]
where \( \dot{\mathbf{r}}_i^- \) and \( \dot{\mathbf{r}}_{iq}^- \) are the known pre-impulse velocities. Let us examine the change in velocities in more detail.

Denote the \( i^{th} \) impulse by \( \lambda_i \mathbf{n}_i \). Expanding \( \delta \dot{\mathbf{r}}_i^\text{ip} \) in terms of the effect of all impulses on the center of mass velocity and the effect of all impulse-induced torques on the angular velocity, we find that  
\[
\delta \dot{\mathbf{v}}_p = M_p^{-1} \sum_{j=0}^{k-1} s_{jp} \lambda_j \mathbf{n}_j.
\]
To compute the change in velocity, we simply sum the effect of all impulses on body \( p \) (let \( s_{jp} \) indicate whether impulse \( j \) acts on body \( p \), and if so capture the correct sign):  
\[
M_p^{-1} \sum_{j=0}^{k-1} s_{jp} \lambda_j \mathbf{n}_j.
\]

To compute the impulse-induced torque, we sum impulses crossed with their point of action relative to the center of mass of body \( p \):  
\[
\delta \mathbf{\omega}_p = I_p^{-1} \sum_{j=0}^{k-1} s_{jp} \lambda_j \mathbf{n}_j \times \mathbf{r}_{jp}^j.
\]
Making these substitutions, we have:  
\[
\dot{\mathbf{r}}_i^\text{ip} = M_p^{-1} \sum_{j=0}^{k-1} s_{jp} \lambda_j \mathbf{n}_j + I_p^{-1} \sum_{j=0}^{k-1} s_{jp} \lambda_j \mathbf{n}_j \times \mathbf{r}_{jp}^j.
\]
Armed with the knowledge of how all impulses change the velocity at the points of contact, we can expand the constraint as:

\[
\dot{g}_i = \dot{n}_i \cdot (\ddot{r}_{ip} - \ddot{r}_{iq}) + \dot{n}_i \cdot (\delta\dot{r}_{ip} - \delta\dot{r}_{iq})
\]

\[
= \sum_{j=0}^{k-1} \lambda_j (s_{jp} \hat{n}_i^T M_p^{-1} \hat{n}_j) + \sum_{j=0}^{k-1} \lambda_j (s_{jp} \hat{n}_i^T I_p^{-1} \dot{r}'_{jp} \times \hat{n}_j \times \dot{r}'_{ip})
\]

\[
- \sum_{j=0}^{k-1} \lambda_j (s_{jq} \hat{n}_i^T M_q^{-1} \hat{n}_j) - \sum_{j=0}^{k-1} \lambda_j (s_{jq} \hat{n}_i^T I_q^{-1} \dot{r}'_{jq} \times \hat{n}_j \times \dot{r}'_{iq})
\]

\[
+ \dot{n}_i \cdot (\ddot{r}_{ip} - \ddot{r}_{iq})
\]

Observe that this constraint is linear in all impulse magnitudes \(\lambda_j\). We can therefore express the contact constraints as a linear function of the impulse magnitudes: \(A\lambda + b\). The \(i^{th}\) entry of this vector gives the post-impulse relative velocity at the \(i^{th}\) contact. The non-penetration constraints are thus:

\[
A\lambda + b \geq 0
\]

Our work is not yet complete, however. As currently formulated, nothing prevents constraint impulses from pulling. Unless we are modeling an adhesive surface, this behavior is undesirable, so we impose the additional constraint that contact forces can only push:

\[
\lambda \geq 0
\]

If we were to stop here, we would have a linear programming problem that we could solve using any number of algorithms (see Chapter 29 of [Cormen et al., 2009]). We need to introduce one final constraint before our work is complete, however. Due to the coupled nature of the constraints, applying impulses along some given constraint direction might obviate the need the need to apply impulses along another. See Figure 5.19. If we do not enforce the fact that no impulses should be applied along breaking contacts, we could potentially apply a response that is too strong and add energy to the system. Thus, we introduce the condition that:

\[
(A\lambda + b)_i \lambda_i = 0 \quad \forall i
\]

What does this condition say? If \(\lambda_i > 0\), then \((A\lambda + b)_i = 0\). That is, if an impulse is acting at some point, the bodies must remain in contact at that point. If \((A\lambda + b)_i > 0\), then \(\lambda_i = 0\).
That is, if bodies are separating, no impulse can act. This celebrated result is known as the Signorini-Fichera Condition. Together, these three conditions on contact constraints can be solved as a linear complimentarity problem.

### 5.9.3.2 Example System

Before proceeding, we will first examine an example problem and its solution.

### 5.9.3.3 Block On Table Edge With Two Contacts

Consider a block sitting on a fixed table with 2 contact points, and with the block’s center of mass positioned beyond the edge of the table. See Figure 5.20. Labeling the rigid body $A$, we can write the constraints on the relative velocity as:

- $\dot{g}_0 = \hat{n}_0 \cdot \hat{r}_{0A} = \hat{n}_0 \cdot \hat{r}_{0A}^+ + \hat{n}_0 \cdot \delta \hat{r}_{0A} \geq 0$

- $\dot{g}_1 = \hat{n}_1 \cdot \hat{r}_{1A} = \hat{n}_1 \cdot \hat{r}_{1A}^- + \hat{n}_1 \cdot \delta \hat{r}_{1A} \geq 0$

Expanding $\delta \hat{r}_{0A}$ and $\delta \hat{r}_{1A}$ we find

\[
\delta \hat{r}_{0A} = \delta \hat{v}_A + \delta \hat{\omega}_A \times \hat{r}_{0A}'
\]

\[
\delta \hat{r}_{1A} = \delta \hat{v}_A + \delta \hat{\omega}_A \times \hat{r}_{1A}'
\]

where primes denote the positions of the contacts relative to the body’s center of mass. $\delta \hat{v}_A$ is the change in the center of mass’ velocity due to both impulses, and $\delta \hat{\omega}_A$ is the change in angular velocity about the center of mass due to both impulses. Expanding $\delta \hat{v}_A$ and $\delta \hat{\omega}_A$ we
\[ \delta v_A = \lambda_0 \hat{n}_0 / M_A + \lambda_1 \hat{n}_1 / M_A \]
\[ \delta \omega_A = r'_{0A} \times \lambda_0 \hat{n}_0 / I_A + r'_{1A} \times \lambda_1 \hat{n}_1 / I_A \]

Substituting these expressions into the constraints, we find
\[
\begin{align*}
\dot{g}_0 &= \hat{n}_0 \cdot \hat{r}_{0A}^- + \lambda_0 \left( \hat{n}_0 \cdot \hat{n}_0 / M_A + \hat{n}_0 \cdot r'_{0A} \times \hat{n}_0 \times r'_{0A} / I_A \right) \\
&\quad + \lambda_1 \left( \hat{n}_0 \cdot \hat{n}_1 / M_A + \hat{n}_0 \cdot r'_{1A} \times \hat{n}_1 \times r'_{1A} / I_A \right) \geq 0 \\
\dot{g}_1 &= \hat{n}_1 \cdot \hat{r}_{1A}^- + \lambda_0 \left( \hat{n}_1 \cdot \hat{n}_0 / M_A + \hat{n}_1 \cdot r'_{0A} \times \hat{n}_0 \times r'_{0A} / I_A \right) \\
&\quad + \lambda_1 \left( \hat{n}_1 \cdot \hat{n}_1 / M_A + \hat{n}_1 \cdot r'_{1A} \times \hat{n}_1 \times r'_{1A} / I_A \right) \geq 0
\end{align*}
\]

or in matrix form
\[
\begin{pmatrix}
\hat{n}_0 \cdot \hat{n}_0 / M_A + \hat{n}_0 \cdot r'_{0A} \times \hat{n}_0 \times r'_{0A} / I_A \\
\hat{n}_1 \cdot \hat{n}_0 / M_A + \hat{n}_1 \cdot r'_{0A} \times \hat{n}_0 \times r'_{0A} / I_A
\end{pmatrix}
\begin{pmatrix}
\lambda_0 \\
\lambda_1
\end{pmatrix}
+ \begin{pmatrix}
\hat{n}_0 \cdot \hat{r}_{0A}^- \\
\hat{n}_1 \cdot \hat{r}_{1A}^-
\end{pmatrix}
\geq \begin{pmatrix}
0 \\
0
\end{pmatrix}
\]

Let us substitute in specific values and examine the solution. Let \( \hat{n}_0 = (0, 1)^T, \hat{n}_1 = (0, 1)^T, \) \( M_A = 4, I_A = 20, r'_{0A} = (-2, -1, 0), r'_{1A} = (-1, -1, 0), \) \( \hat{r}_{0A}^- = (0, -2, 0), \) and \( \hat{r}_{1A}^- = (0, -2, 0). \)

This gives the system:
\[
\begin{pmatrix}
13/20 & 9/20 \\
9/20 & 7/20
\end{pmatrix}
\begin{pmatrix}
\lambda_0 \\
\lambda_1
\end{pmatrix}
+ \begin{pmatrix}
-2 \\
-2
\end{pmatrix}
\geq \begin{pmatrix}
0 \\
0
\end{pmatrix}
\]

which is solved by \( \lambda_0 = 0 \) and \( \lambda_1 \approx 5.7143 \) (as a sanity check, verify that these values satisfy the three constraints in the LCP). What does this mean? \( \lambda_0 = 0 \) implies that the leftmost impulse is not active and the contact is breaking. \( \lambda_1 \approx 5.7143 \) implies that the second point remains in contact with the table, and that an impulse is acting to prevent interpenetration.

The net effect is that the block will pivot about the end of the table, as expected.

### 5.9.4 Redundant Constraints

While our formulation is enough to obtain a correct solution, it does not pin down a unique solution. For example, Figure 5.21 illustrates a table with three legs resting on the ground.
Figure 5.21: Static indeterminacy: A table under gravity in two dimensions with three contacts. Observe that multiple sets of impulses give the same solution - we could apply only the large impulse to the center leg, or the two smaller impulses to the outer legs.

There are a (uncountably infinite) number of solutions to this contact problem as we have phrased it. We could apply a single impulse of magnitude $I$ to the center leg and no impulses to the side legs, we could apply two impulses of magnitude $I/2$ to the side legs, or we could apply any combination of (pushing) impulses with total magnitude $I$ that induce no torque.

What does this redundancy mean in linear algebra terminology? For the three legged table, the matrix $A$ is singular. That is, $A\lambda + b = 0$ has multiple solutions. Many of the standard LCP algorithms, unfortunately, will fail when presented with a singular matrix. While one can construct a number of interesting problems that do not have this singular matrix, if we desire a truly robust method we will have to work a little harder. It is important to stress that this is a numerical issue. Conceptually (numerics aside), even though $\lambda$ is not uniquely defined, the final velocities are uniquely defined. Luckily, this problem can be solved by a number of numerical methods, some of which are more robust to this redundancy than the LCP. We will repose this problem as a minimization, but first let us formalize some of the concepts that we explored above, using a notation that applies equally to the LCP and minimization viewpoints.

5.9.5 More General Notation

The notation employed thus far has been traditionally used in the graphics literature (see Baraff, 1989a). While the derivations so far have been straightforward in that they follow immediately from writing down the physics of rigid bodies and ‘turning the crank’, they can be a tad unwieldy.

Before rephrasing the contact problem to address the singular matrix $A$, let us take a
moment to construct a more general notation than the one employed above. Rewriting the notation will have a few advantages. First, the more general notation will make our results easier to carry over to other settings (e.g. simulation of deformable bodies). Second, the more general notation will make higher-level algebraic manipulations easier, and makes it easier to see higher-level properties of the system, such as the symmetry of $A$.

5.9.5.1 Notation

Take each rigid body’s reduced or generalized coordinate representation (center of mass position, orientation), and concatenate them into a single column vector $q$. Similarly, concatenate the rate of change of the reduced coordinates into a single column vector $\dot{q}$. As an example, for our 2D representation or rigid bodies, $q \in \mathbb{R}^3N$ and $\dot{q} \in \mathbb{R}^3N$, where $N$ is the number of rigid bodies. For 3 rigid bodies these vectors look like:

$$q = \begin{pmatrix} X_0^0 & X_0^1 & \theta^0 & X_1^0 & X_1^1 & \theta^1 & X_2^0 & X_2^1 & \theta^2 \end{pmatrix}$$

$$\dot{q} = \begin{pmatrix} V_0^0 & V_0^1 & \omega^0 & V_1^0 & V_1^1 & \omega^1 & V_2^0 & V_2^1 & \omega^2 \end{pmatrix}$$

When applying forces and impulses to rigid bodies, we need some way to compute world-space positions and velocities from reduced positions and velocities. Let $r_i(q)$ be a function that computes the world space position of the $i^{th}$ point from the reduced representation. For example, if the seventh particle is attached to the first rigid body, in our representation this function takes the form (tilde denotes body space):

$$r_7(q) = R(\theta^1)\tilde{r}_7 + X^1$$

To compute the world-space velocity of the $i^{th}$ particle, we simply employ the chain rule. That is, $\dot{r}_i(q) = \nabla r_i \dot{q}$. Continuing with the above example, $\nabla r_7 \in \mathbb{R}^{2\times 9}$ is given by:

$$\nabla r_7 = \begin{pmatrix} 0 & 0 & 0 & 1 & -\sin \theta^1 \tilde{r}_{7x} - \cos \theta^1 \tilde{r}_{7y} & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & \cos \theta^1 \tilde{r}_{7x} - \sin \theta^1 \tilde{r}_{7y} & 0 & 0 & 0 \end{pmatrix}$$

Convince yourself that the multiplication $\nabla r_i \dot{q}$ gives the same velocity we derived in Theme III Milestone I.

Consider now a set of contacts $C$. For some contact $k \in C$, the contact occurs between two points $r_i \in \mathbb{R}^2$ and $r_j \in \mathbb{R}^2$. Let the normal for this contact, $\hat{n}_k \in \mathbb{R}^2$, point from $i$ to $j$. We can
thus express the relative velocity at this contact as $\hat{n}_k^T(\dot{r}_i - \dot{r}_j) = \hat{n}_k^T(\nabla r_i - \nabla r_j)\dot{q} \equiv \hat{n}_k^T \Gamma_k \dot{q}$. For our simulations, $\Gamma_k \in \mathbb{R}^{2 \times 3N}$, and contains 2 non-zero $\mathbb{R}^{2 \times 3}$ blocks. Continuing our example of three rigid bodies, if the 3rd contact is between the 7th point (on body 1) and the 9th point (on body 2), $\Gamma_3$ is given by:

$$
\begin{pmatrix}
0 & 0 & 0 & 1 & 0 & -\sin \theta_1 \tilde{r}_{7x} - \cos \theta_1 \tilde{r}_{7y} & -1 & 0 & \sin \theta_2 \tilde{r}_{9x} + \cos \theta_2 \tilde{r}_{9y} \\
0 & 0 & 0 & 1 & \cos \theta_1 \tilde{r}_{7x} - \sin \theta_1 \tilde{r}_{7y} & 0 & -1 & -\cos \theta_2 \tilde{r}_{9x} + \sin \theta_2 \tilde{r}_{9y}
\end{pmatrix}
$$

If we want to apply an equal and opposite impulse $y$ to points $i$ and $j$ at contact $k$, the resulting impulse on the reduced coordinates is given by $\Gamma_k^T y$. Similarly, we can map the contact normals to their reduced coordinate counterparts by $\eta_k = \Gamma_k^T \hat{n}_k \in \mathbb{R}^{3N}$. Finally, concatenating these normals into one matrix, we have $N = \begin{pmatrix} \eta_0 & \eta_1 & \ldots & \eta_{|C|-1} \end{pmatrix} \in \mathbb{R}^{3N \times |C|}$.

5.9.5.2 A Revisited

Let us revisit the construction of the system $A\lambda + b$ for the LCP. The post-impulse relative velocity at all contact points in world-space can be expressed in terms of the reduced coordinates as $N^T \dot{q}^+$. Mapping the impulse magnitudes in world-space to the reduced coordinate space gives $N\lambda$, and the corresponding changes in velocities are expressed as $M^{-1}N\lambda$. Thus, we find that $N^T \dot{q}^+ = N^T (\dot{q}^- + M^{-1}N\lambda) = N^T M^{-1}N\lambda + N^T \dot{q}^- \geq 0$. That is, $A = N^T M^{-1}N$ and $b = N^T \dot{q}^-$. The symmetry of $A$ now follows immediately from the symmetry of $M$ (take the transpose of $N^T M^{-1}N$ and see what you get).

5.9.6 Velocity-Projection for Inelastic Contact

When solving the LCP, we encountered problems with a singular matrix that arose due to redundant contact directions. To sidestep this complication, we can rephrase the inelastic contact problem as the minimization of change in kinetic energy subject to non-penetration constraints. See Figure 5.22 for an illustration of this method. The result is a convex quadratic-program, an optimization problem for which algorithms exist that are robust in the face of redundant constraints (see [Goldfarb and Idnani, 1983]).

More precisely, the post-collision velocity is given by the minimizer of $\delta T = \frac{1}{2}(v - \dot{q}^-)^T M(v - \dot{q}^-)$ subject to $N^T v \geq 0$. Expanding this multiplication, we find that $\delta T = \frac{1}{2}v^T M v - \frac{1}{2}v^T M \dot{q}^- - \frac{1}{2}(\dot{q}^-)^T M v - \frac{1}{2}(\dot{q}^-)^T M \dot{q}^- = \frac{1}{2}v^T M v - v^T M \dot{q}^- - \frac{1}{2}(\dot{q}^-)^T M \dot{q}^-$ where we have used
Figure 5.22: Velocity projection: On the left, unit-mass particles moving in configuration space constrained to lie above the horizontal. On the right, the same particles in velocity space and the constraint impulses. Post-impulse velocities are denoted by stars. Notice that the post-impulse velocities are given by the closest point in the admissible region.

the symmetry of $M$ to combine the cross terms. We can drop the constant term $-\frac{1}{2}(\dot{q}^-)^T M \dot{q}^-$, as constant offsets will not change the minimum of the function. Thus, the post-collision velocities are given by:

$$\dot{q}^+ = \text{argmin}_v \left( \frac{1}{2} v^T M v - v^T M \dot{q}^- : N^T v \geq 0 \right)$$

Observe that we never explicitly compute or apply impulses with this formulation.

5.9.6.1 Intuitive Connection to LCP

This formulation bears very little resemblance to the LCP, but they are equivalent formulations. We could demonstrate that the LCP is the optimality condition (Karush-Kuhn-Tucker, or KKT condition) for this minimization, but for our purposes it will suffice to develop some intuition. Referring to Figure 5.22, we see that the complementarity condition is indeed satisfied; impulses are only active for velocities that are bringing particles into the inadmissible region. Further, notice that impulses only act in the positive constraint gradient direction, that is they are only pushing. Finally, observe that all post-impulse relative velocities are separating or zero. Thus, all of the content of the LCP formulation is reflected in the velocity-projection solution.
5.9.7 Required Features

5.9.7.1 Rigid-Body Rigid-Body All-Pairs Detection

To detect collisions, please complete the `detectCollisions` method of `RigidBodyAllPairsCollisionDetector.cpp`. As input this method takes a vector of all rigid bodies in the simulation, and as output constructs a set of template type `RigidBodyCollision` containing all collisions in the system. For each pair of rigid bodies, detect any vertex-edge collisions. A vertex and an edge are considered colliding if the distance between the two is less than (keep this inequality strict!) the sum of the radii of each body. You should be able to adapt this code from Theme II. When inserting a `RigidBodyCollision` into the collision set, you will have to compute:

1. The index of the first rigid body of the collision ($i$ in the constructor)
2. The index of the second rigid body of the collision ($j$ in the constructor)
3. The vector from the center of mass of the first body to the point of contact ($r_0$ in the constructor)
4. The vector from the center of mass of the second body to the point of contact ($r_1$ in the constructor)
5. The unit-length contact normal ($nhat$ in the constructor)

5.9.7.2 LCP

Please complete the `resolveCollisions` method of `RigidBodyLCPCollisionResolver.cpp`. As input this method takes both a vector of all rigid bodies and a set of all collisions between rigid bodies, and as output modifies the velocities and angular velocities of the rigid bodies so as to prevent penetration.

You will have to compute the matrix $A$ and the vector $b$ described in Sections 5.9.3 and 5.9.5.2. We have provided a function `lcutils::solveLCPwithODE` that takes as inputs $A$ and $b$ and returns a vector $\lambda$ of impulse magnitudes (please see the code for an example). You will then have to apply the resulting impulses to the rigid bodies.
We provide the expected values for $A$ and $b$ in the scene files `collisionslcp/test01.xml` and `collisionslcp/test03.xml`. This code uses the implementation of Dantzig’s method from the open source rigid body engine ODE [Smith, 2006].

*Note on fixed rigid bodies:* Impulses have no effect on fixed rigid bodies. Therefore, to handle fixed rigid bodies in the response, when you are constructing $A$ ignore any terms corresponding to an impulses' effect on a fixed body.

### 5.9.7.3 Velocity-Projection

Please complete the `resolveCollisions` method of `RigidBodyVelocityProjectionCollisionResolver`. As input this method takes both a vector of all rigid bodies and a set of all collisions between rigid bodies, and as output modifies the velocities and angular velocities of the rigid bodies so as to prevent penetration.

You will have to compute the mass matrix $M$, the product $-M\dot{q}$, and the generalized constraint matrix $N$ described in Sections 5.9.5 and 5.9.6. We have provided a function `solve_quadprog` that takes these quantities as inputs and returns the solution to the QP. Please see the code for an example.

We have provided the expected values for $M$, $-M\dot{q}$, and $N$ in the scene files `collisionsvelocityprojection/test01.xml` and `collisionsvelocityprojection/test05.xml`. This code uses the open-source QuadProg++ [Gaspero and Moyer, 2009] implementation of the Goldfarb-Idnani algorithm.

*Note on fixed rigid bodies:* The easiest way to handle fixed bodies in the solve is to simply not expose those bodies’ degrees of freedom to the QP solver. That is, shrink $M$, $M\dot{q}$, and $N$ to not include fixed degrees of freedom, and do not add fixed bodies’ contributions to $N$.

### 5.9.8 Future Work

We have only begun to scratch the surface of the vast body of literature on rigid body contact. In particular, we have not addressed the issue of friction. Interested readers should refer to the recent work of Kaufman et al. [Kaufman et al., 2008a] as a starting point. Having completed Milestones I and II, you are in a great position to tackle this paper.
5.10 Fluid Simulation: Theme IV

5.10.1 Introduction

In this theme, we will explore fluid simulation for animation. In particular, we will examine a method popular in the graphics community known as *stable fluids*.

The time evolution of a fluid is typically modeled with the *Navier-Stokes* equations. The *Navier-Stokes* equations are a collection of nonlinear partial differential equations that, when solved, yield the velocity field that governs the motion of a fluid. The *Navier-Stokes* equations can be written in a number of forms – we are particularly interested in the viscous and incompressible case, as most everyday fluids of visual interest are viscous and incompressible. *Stable fluids* handles this particular case.

5.10.2 Grading

This milestone is structured differently than previous milestones. In lieu of an oracle whose output you must match, your program will be graded based on if it appears visually correct, and if it runs interactively. You will also be given greater leeway in implementing the fluid simulation, although we are still providing you with rendering and user interface code. With that said, we strongly urge you to start early and stop by office hours if you have any questions about your progress.

5.10.3 Implementation

Please read and implement the paper *Stable Fluids* by Jos Stam [Stam, 1999]. For further implementation details, please see Jos Stam’s paper *Real-time fluid dynamics for games* [Stam, 2003]. We have linked to these papers on the course wiki. *Stable Fluids* provides a nice description of the method, while *Real-time fluid dynamics for games* fleshes out a self-contained implementation. We suggest that you read *Stable Fluids* for a high-level overview of the method, and then mirror the implementation in *Real-time fluid dynamics for games*. 
5.10.4 Starter Code

We have posted starter code to the course wiki. Please implement your solution using the\n\textit{StableFluidsSim} class. Again, we have provided both rendering and user-interface code.

5.10.5 User-Interface Controls

Right-clicking and dragging deposits ‘marker’ fluid in the simulation grid. Left-clicking and\ndragging applies a force to the velocity field. The plus and minus keys switch between colormaps\nfor visualizing the flow. The r key resets the simulation.

5.10.6 Future Work

If you are interested in learning more about fluid simulation for animation, we recommend\nthe SIGGRAPH 2007 Fluid Simulation Course Notes by Robert Bridson and Matthias Müller-\nFischer \cite{Bridson2007}.

5.11 Final Creative Scene: Theme V

5.11.1 Creative Scene

For this milestone, please include a scene of your design that best shows off what you have\nlearned this semester in class. This scene will count for 100\% (all) of your grade for the\nassignment. Based on the quality of your scene, you will have the opportunity to earn up to\n15\% extra credit. Your scene will be judged by a secret committee of top scientists using the\nhighly refined criteria of:

1. How well the scene shows off all of the simulation techniques you have learned this\n   semester.

2. Aesthetic considerations. The more beautiful, the better.

3. Originality. Implementing a feature that was discussed in class, but wasn’t included in\n   any of the milestones, will be viewed very favorably.
Top examples will be posted to the course wiki. To aid you in constructing more complex scenes, a collection of python scripts have been posted to the wiki.

To submit this scene, simply place it in the CreativeScene directory of your submission.

**Important Note:** Please name your scene file `t5_youruni_scenenumber.xml` where `youruni` is your uni. For example, if I submitted two scenes I would name them `t5_bds2114_1.xml` and `t5_bds2114_2.xml`. **If your scenes do not follow this convention, you will receive no credit.**

**Important Note:** Creative scenes **must** include a title as their description in the XML file.

Good luck on this final assignment!

### 5.12 Conclusion

In the first semester of this course offering, we developed themes that addressed mass-spring systems, collisions, rigid body simulation, and fluid simulation. With our autograding oracle, students were able to make fast progress through this material. In subsequent course offerings, we have found that the structure we have presented here is easily extended to other topics, including volumetric elastica, thin shell and cloth simulation, elastic rod and hair simulation, and control of physical systems. We are encouraged by the generality of this oracle-based course structure, and we are excited to explore the full extent of topics that can be treated from this framework.
Chapter 6

Conclusions and Future Work

We have addressed the simulation of granular materials from two perspectives: discrete particle simulations with provable guarantees, and hybrid continuum-discrete simulations that inherit benefits from both approaches. Our contributions to discrete particle simulation include: (i) the first non-smooth collision model that is able to simultaneously respect five key physical properties; (ii) a solution to the problem of inelastic collapse in methods for non-smooth, impulse propagation, allowing us to run simulations with confidence that they will terminate; and (iii) a time integrator that preserves the five physical properties of our collision model across discrete time-steps, allowing us to run simulations at coarse temporal resolutions while retaining confidence in the result. These developments result in an algorithm that requires minimal parameter tuning, and we are able to numerically replicate lab experiments by simply duplicating the reported parameters. We are excited both by the prospect of testing our method in new experimental domains, and by the potential for running large scale, automated numerical tests to discover unreported phenomena. We are also excited to further probe the questions our work raises with respect to termination of these models, with initial work that builds off of our approach already showing promise [Vouga et al., 2017].

While discrete particle models are accurate, powerful, and predictive, they are a fundamentally fine grained tool. Granular materials are multi-scale, and bulk regions can behave as a continuum. Towards bridging this gap, we have presented a method to run hybrid simulations with a discrete model coupled to a continuum model. Building this hybrid model has led to three core contributions: (i) a theoretical framework through which to view the coupling of
continuum and discrete granular models, which reveals a family of possible numerical integrators; (ii) building off this framework, we demonstrate a method that is able to couple a discrete element simulation to a material point simulation with no linear solve; (iii) a method to dynamically track regions suitable for a homogenized continuum treatment and to convert between continuum and discrete treatments as needed. We are excited at the prospect of expanding our range of validations, especially against 3D, lab-scale experiments. As detailed in Section 3.6, the framework we have established can lead to an entire family of integrators for hybrid, granular systems, and we are excited to develop these methods. Finally, we are excited to develop a deeper understanding of where our technique for identifying continuum regions breaks down, which we hope will lead to a method that is able to capture even more phenomena of importance, including shear banding.

Finally, to aid in teaching the tools of physically based animation to students of computer science, we have contributed a novel system for vetting the correctness of new implementations of numerical algorithms. The course that we have structured around this autograding numerical oracle has proven successful at Columbia University, with larger sustained course sizes and student retention rates over multiple semesters compared to the previous course design. Beyond expanding the scope of the course materials, we are interested in possible applications of our numerical autograder outside of education. From a software engineering and computer science perspective, the numerical oracle fills a role similar to that of test-driven development: when optimizing, refactoring, or otherwise making major changes to an existing numerical integrator, a numerical oracle could easily detect regressions. The numerical oracle could also enable easier direct comparisons on systems where small numerical perturbations can produce massive changes in trajectories (quite common with rigid multi-body simulations, for instance). By always restarting each time-step from the oracle’s state, the new algorithm would always evaluate the same state as the oracle, eliminating any differences that stem from the diverging trajectories. We are excited about the potential benefits that this technology can yield.
Bibliography


[HSL, 2001] HSL. A collection of Fortran codes for large scale scientific computation. 

[Hsu and Keyser, 2010] Shu-Wei Hsu and John Keyser. Piles of objects. ACM Trans. Graph.,

[Ihmsen et al., 2013] Markus Ihmsen, Arthur Wahl, and Matthias Teschner. A lagrangian
   framework for simulating granular material with high detail. Computers & Graphics,


   of rigid body collections. In Proceedings of Contact Mechanics International Symposium,


[Jiang et al., 2016] Chenfanfu Jiang, Craig Schroeder, Joseph Teran, Alexey Stomakhin, and
   Andrew Selle. The material point method for simulating continuum materials. In ACM
   ACM.


[Jop et al., 2006] Pierre Jop, Yoël Forterre, and Olivier Pouliquen. A constitutive law for dense


Appendix A

Core Desiderata under Interpolation

A.1 Preservation of Desiderata

We prove that our proposed Generalized Restitution model produces feasible post-impact velocities and continues to satisfy our five core identified desiderata.

A.1.1 Feasibility of the Solution

A feasible post-impact velocity satisfies \( G(q)^T \dot{q}^+ \geq 0 \).

**Theorem.** Interpolation yields a feasible post-impact velocity for all coefficients of restitution \( c_r \in [0,1] \).

**Proof.** Computing the post-impact relative velocity, we obtain:

\[
G^T \dot{q}^+ = (1 - c_r) G^T \dot{q}_0^+ + c_r G^T \dot{q}_1^+
\]

By construction the LCP model guarantees that \( G^T \dot{q}_0^+ \geq 0 \). Similarly, upon termination the GR model guarantees that \( G^T \dot{q}_1^+ \geq 0 \). Each term in this sum is non-negative. Therefore the interpolation yields a feasible velocity.

A.1.2 Conservation of Momentum

We begin with the observation that interpolating two post-impact velocities is equivalent to interpolating the corresponding impulses.
**Lemma.** Interpolating \( \dot{q}_0^+ \) and \( \dot{q}_1^+ \) is equivalent to interpolating \( \lambda_0 \) and \( \lambda_1 \).

**Proof.**

\[
\dot{q}^+ = (1 - c_r) \dot{q}_0^+ + c_r \dot{q}_1^+ \\
= (1 - c_r) (\dot{q}^- + M^{-1} G \lambda_0) + c_r (\dot{q}^- + M^{-1} G \lambda_1) \\
= \dot{q}^- + M^{-1} G ((1 - c_r) \lambda_0 + c_r \lambda_1)
\]

Therefore the net impulse magnitude is \( \lambda = (1 - c_r) \lambda_0 + c_r \lambda_1 \).

**Theorem.** Interpolation conserves momentum.

**Proof.** The generalized normals, by construction, conserve momentum and angular momentum, therefore \( G \lambda \) exerts a momentum conserving impulse on the system for any given set of magnitudes \( \lambda \). The interpolated response thus conserves momentum.

**A.1.3 One-Sided Impulses**

A one-sided impulse satisfies \( \lambda \geq 0 \).

**Theorem.** Interpolation produces one-sided impulses for all \( c_r \in [0, 1] \).

**Proof.** Given two sets of one-sided impulses \( \lambda_0 \geq 0 \) and \( \lambda_1 \geq 0 \), the sum \( (1 - c_r) \lambda_0 + c_r \lambda_1 \geq 0 \) is also one-sided.

**A.1.4 Bounded Kinetic Energy**

The post-impact kinetic energy is given by

\[
T (c_r) = \frac{1}{2} \left( (1 - c_r) \dot{q}_0^+ + c_r \dot{q}_1^+ \right)^T M \left( (1 - c_r) \dot{q}_0^+ + c_r \dot{q}_1^+ \right)
\]

**Theorem.** Interpolating post-impact velocities from an inelastic and from an elastic response yields a post-impact kinetic energy bounded by that of elastic response.

**Proof.** The kinetic energy is quadratic in \( c_r \) and \( T (0) < T (1) \). Therefore, if the second derivative of the energy with respect to \( c_r \) is positive, the energy can never exceed that of the elastic response when \( c_r \in [0, 1] \). Computing the second derivative, we find that

\[
\frac{\partial^2 T}{\partial c_r^2} = (\dot{q}_1^+ - \dot{q}_0^+)^T M (\dot{q}_1^+ - \dot{q}_0^+).
\]
M is positive definite, which implies that the second derivative is positive. Therefore, the post-impact kinetic energy is bounded by that of the elastic response.

A.1.5 Preservation of Symmetry

The interpolation model does not act on the configuration $q$ of the system, therefore we only consider its effect on the system’s velocity $\dot{q}$.

**Theorem.** Interpolation preserves symmetry.

**Proof.** Let $S(q) = q$ define a (potentially nonlinear) symmetry in the system’s configuration. This map operates linearly on the velocity as $\nabla S(q) \dot{q} = \dot{q}$. Given two velocities that respect this symmetry, we find for the interpolant:

$$
\nabla S \dot{q}^+ = \nabla S \left( (1 - c_r) \dot{q}_0^+ + c_r \dot{q}_1^+ \right) \\
= (1 - c_r) \nabla S \dot{q}_0^+ + c_r \nabla S \dot{q}_1^+ \\
= (1 - c_r) \dot{q}_0^+ + c_r \dot{q}_1^+ \\
= \dot{q}^+
$$

Therefore, the interpolated response preserves symmetry.

A.1.6 Break-Away

**Theorem.** If a post-impact velocity satisfies $\nabla g(q)^T \dot{q}_1^+ > 0$ under GR, then the interpolated post-impact velocity satisfies $\nabla g(q)^T \dot{q}^+ > 0$.

**Proof.** Under interpolation with the inelastic LCP response $\nabla g^T \dot{q}_0^+ \geq 0$, we find that

$$
\nabla g^T \dot{q}^+ = (1 - c_r) \nabla g^T \dot{q}_0^+ + c_r \nabla g^T \dot{q}_1^+ > 0
$$

for all $c_r \in (0, 1]$.