Inverse Methods for Design and Simulation with Particle Systems

by

Erik Steven Strand

S.B., University of Chicago (2012)

Submitted to the Program in Media Arts and Sciences
in partial fulfillment of the requirements for the degree of

Master of Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

September 2020

Author .....
Program in Media Arts and Sciences
August 17, 2020

Certified by .....
Prof. Neil Gershenfeld
Director, MIT Center for Bits and Atoms
Thesis Supervisor

Accepted by .....
Prof. Tod Machover
Academic Head, Program in Media Arts and Sciences

Inverse Methods for Design and Simulation with Particle Systems

by

Erik Steven Strand

Submitted to the Program in Media Arts and Sciences
on August 17, 2020, in partial fulfillment of the
requirements for the degree of
Master of Science

Abstract

Over the last several decades, computer aided design (CAD) and numeric simulation software have become ubiquitous in engineering research and practice. Despite this, tools that close the loop between design and simulation — such as optimizing a design based on simulated performance, or searching over simulation parameters to identify where a design functions as intended — remain highly specialized and relatively underutilized.

This thesis charts a path to greater adoption of inverse methods over physical simulation. I demonstrate a portable, high performance simulation tool based on dynamic mesh free particle systems, as well as a generic framework for implementing algorithmically modifiable design languages. I discuss best practices in the use of optimization algorithms and the development of objective functions for simulation based inverse methods. Finally, I present two applications made possible by these tools: optimization of gear tooth profiles, and automated construction of coarse grained models for simulation of plastic deformation of polymers.

Thesis Supervisor: Prof. Neil Gershenfeld
Title: Director, MIT Center for Bits and Atoms

Inverse Methods for Design and Simulation with Particle Systems

by

Erik Steven Strand

This thesis has been reviewed and approved by the following committee members

Neil Gershenfeld .....
Director, Center for Bits and Atoms
Professor, Media Arts and Sciences
MIT

Wojciech Matusik .....
Director, Computational Fabrication Group
Professor, Electrical Engineering and Computer Science
MIT

John Hart .....
Director, Mechanosynthesis Group
Professor, Mechanical Engineering
MIT

Acknowledgments

Neil, thank you for making and steering the Center for Bits and Atoms. The rate at which I pick up new ideas and skills in this lab far outpaces anything I've experienced before, and anything I reasonably expected would be possible.

To my readers, Wojciech and John, thank you for taking the time to provide your input. I look forward to more collaboration as I work toward my PhD.

To my prior academic mentors, particularly Christian and Scott, thank you for trusting me to do real research before I was remotely qualified.

To the students and staff of CBA, thank you for the jokes, stories, insights, and escapades. To Jake, extra thanks for suggesting that I apply to grad school in the first place.

To the people of Otherlab and FabLight, my career jump from consulting to making laser cutters was a watershed moment that ultimately led me where I am today. Thanks for showing me the path.

Wes, Michael, Amir, Nick, Jeremy, and all the other Plethorans, thanks for the backyard barbecues and honing my coding and fabrication chops.

To my fellow Jazztronauts, Max and Michael, thanks for all the funky grooves, sizzling swing, and lurchy backbeats. I look forward to playing again soon. Felisa, thanks for your kindness and lomo saltado. To everyone at PianoFight, thanks for making a home away from home.

To the talented musicians of AMP, FJE, and Rambax, thank you for keeping music in my life, no matter how overbooked my schedule becomes.

Chris, Natasha, and Kelly, thanks for the conversations, cooking, and encouragement.

Mikhail, Sam, and Hannah, thanks for making the East Coast feel like home, and, with Jay, for the formative experiences in distant cities, mountains, and fjords. Jenni, Andy, Jaylib, and Michael, thanks for the lively debates and constant learning, from dorm rooms to dinner parties (or hot tubs). Will, thanks for sharing the eccentric journey from industry to academia.

Hanny, thank you for providing support, sharing the sublime, and trying all my weird mixed drinks. To Niku, thanks for all the wiggles.

Finally a huge thank you to all my family. Mom, Dad, and Sarah, thank you for cheering me on, listening to me, and providing perspective.

Contents

Chapter 1

Introduction

1.1 Background

A typical engineering development cycle can be divided into four steps. First, a design is drawn, typically in a computer aided design (CAD) system. Second, the design is simulated, using a physics package. Third, the results of the simulation are used to evaluate the performance of the design. Finally, the evaluation informs an assessment of how the design may be improved. These steps are repeated, in an effort to produce a better design with each cycle.

This process is often guided manually. But when all four steps are performed algorithmically, the whole cycle can be automated. This opens the door to *design optimization*, i.e. the automated improvement of a design using computational techniques, as well as *inverse design*, i.e. the use of design optimization techniques to completely specify (rather than modify) a design. These sorts of techniques are examples of *inverse methods* over simulation, in which initial conditions and simulation parameters that achieve a desired outcome are identified, despite the fact that no directly computable method exists for working backwards to a solution.

This process is most commonly used to optimize a design's geometry for performance. An airfoil, for example, can have its profile manipulated to achieve a desired lift to drag ratio [57]. But the technique is more general than such an example reveals. The physical design of a robot arm may be fixed, but one can still search for optimal

control strategies for various tasks [43]. Even the parameters of the simulation itself can be targeted by an inverse method. This technique can be used, for example, to develop approximate realtime emulations of computationally expensive simulations [35].

Despite these successes, it is my view that inverse methods over simulation are both underdeveloped and underutilized. There are many tools that address specific applications in inverse design, such those cited above, or the others that will be discussed in the proceeding section. On the whole, they tend to be highly specialized, are quite often cumbersome, and are used far less than CAD or simulation tools are on their own. Inverse tools for the optimization of simulations themselves are comparatively few and far between.

I believe that inverse methods over simulation can be approached in a much more general manner. Doing so requires careful consideration of what simulations we use, how we represent what we want to improve, which optimization algorithms we apply, and how we evaluate performance along the way. This thesis addresses all of these issues. Specifically, I present a new simulation tool and a new design representation framework, then discuss the metrics that one should keep in mind when selecting an optimization algorithm to use with them. I demonstrate the use of these tools in a design optimization problem, namely the optimization of gear tooth profiles, and a simulation development problem, namely the construction of coarse grained models for plastic deformation of polymers. Finally, I discuss some common themes that arise in the development of the performance evaluation metrics that guide optimization.

1.2 Prior Art

Many different disciplines have developed and deployed tools for inverse problems over simulation. Because these tools tend to be highly specialized, a survey of them is a survey of many narrowly applied techniques rather than a few general ones. For this reason a high level overview of the literature is the best I can hope to provide in this document. Specifically, I will focus on the use of simulation based inverse

methods across engineering, computer graphics, architecture, and machine learning.

One of the earliest — and now among the most widely used — simulation based inverse methods is topology optimization. Topology optimization is most commonly used to optimize the distribution of material in truss style structures, minimizing compliance for a given amount of material. It assumes a voxel representation of the design, and almost always relies on a static finite element method (FEM) simulation [85]. It traces its roots to analytic studies of literal truss structures performed throughout the twentieth century, but was first formulated as a numeric inverse method in the late 1980s by Bensøe and collaborators [10]. It has since been scaled up to and beyond a billion voxels [1]. It is incorporated into a variety of common CAD applications.

Another major example of inverse method in engineering is the field of multidisciplinary design optimization (MDO). Its purpose is to enable design optimization for problems that span multiple domains of physics [63]. However this often comes at a high cost in complexity, and across other dimensions it is as limited as topology optimization. In particular, though dynamic MDO has been investigated, it remains an underdeveloped capability and most MDO frameworks still assume static or quasi-static problem formulations [6]. While MDO tools exist for a wide variety of problems, each one tends to be hyper-specialized. Finally, MDO tools often only aim to optimize particular aspects of a design, thus falling short of full inverse design.

A third example, from farther afield, comes from very large scale integration (VLSI) for digital circuits. Mead and Conway's classic text on the subject emphasizes the use of design rules and hierarchy [64], if not strictly describing an inverse method. But today, automated design is commonly used for placing and routing, which does rely on some simple objective functions and can be integrated with simulations of stray capacitance. These tools have enabled integrated circuits to become the single most complex objects manufactured by humans — so they are worthy of our consideration even if they do not embody inverse methods as much as other examples.

The most prominent early examples of inverse methods in computer graphics are the studies performed by Karl Sims in the 1990s. Most notably, he used genetic algorithms to evolve virtual creatures, based on a variety of objective functions selected

to encourage walking, jumping, swimming, and other behaviors [88]. These studies have inspired many replications. Sims' methodology is much more general than that of most engineering inverse design tools, and thus not coincidentally is among the closest to that presented in this thesis. I seek to apply such techniques to more practical engineering problems.

A recent series of papers from MIT's Computational Fabrication Group demonstrate gradient based optimization within a variety of simulation environments [41, 39]. These papers are the most closely aligned with the goals and methods of this thesis than any others I have encountered.

Architects have developed many inverse design tools, which more commonly aim to aid design space exploration more so than select a single optimal design. This is known as generative design, and these papers frequently reference biological inspiration, [79, 55, 73]. Optimization is often not mentioned. Even in problems where the ideal outcome would be the selection of a single optimal design, such as those considered in this thesis, it is often valuable to consider a range of solutions that demonstrate alternative strategies or design principles. So these studies are more relevant for strict engineering purposes than one might imagine.

The machine learning field, and reinforcement learning in particular, has recently produced a number of impressive results based on inverse methods [72, 44, 82]. These typically optimize a policy or control law represented as a deep neural network, based on an objective function defined within in a simulation. Through some clever algebra, statistical sampling, and judicious use of surrogate models, these methods usually manage to optimize the neural network with gradient descent without needing to differentiate through the whole simulation. This reduces their generality, and in particular rules out the specific applications considered in this thesis, but these methods are still among the most general and powerful encountered in the literature.

Neural networks have also been used for inverse methods that modify the simulations themselves. The first instances of these techniques were used to develop physically realistic looking animations without needing to run computationally expensive simulations [35]. More recently, researchers have investigated the development of sim-

ulations based on data, typically relying on deep neural networks that process graph data [9, 13]. This is now a very active research area, with applications in specific fields such as robotics [58, 3, 2] and more general simulation and control [81, 80].

1.3 Morphogenesis and Evolution

In contrast to the fragmented world of specialized inverse methods devised by humans, it is informative to seek inspiration in biology. In particular, the natural world does contain a single inverse method of astounding power and generality: biological morphogenesis and evolution. These processes provide a primary inspiration for the development of more general inverse methods over physical processes, and the most compelling demonstration of what could be possible.

Morphogenesis is the process by which biological organisms develop their shapes. We can think of it as a fabrication system, in which DNA is the input, the molecular processes of transcription and translation are the fabrication methods, and our bodies are the outputs. In these terms morphogenesis may not sound so unfamiliar to an engineer, but this biological system is unlike any human-made fabrication system in many fundamental ways.

First, though DNA determines the shape of the organism it describes, it doesn't represent the organism's final geometry explicitly. Instead, DNA directs a developmental program that ultimately produces the organism's shape. In this sense it's more akin to source code than a blueprint. This is important because it enables compression; the genome contains billions of base pairs, but these ultimately guide the arrangement of many trillions of cells.

Second, the developmental program that executes DNA (i.e. transcription of DNA into mRNA, and translation of mRNA into proteins by ribosomes) is distributed. All somatic cells receive a complete copy of the genome, but they differentiate and divide based on signals in their immediate environment, without any external or global coordination. This enables exponential growth, which is essential in order to realize the massive dynamic range of the body: amino acids are less than a nanometer long,

yet our bodies are meter scale, a difference of 10 orders of magnitude.

The structure of the developmental program also facilitates repetition, recursion, and hierarchy, since a particular subprogram can be run multiple times (e.g. to generate two arms) or cede control to a different subprogram (e.g. to grow a hand at the end of each). In fact, expression of an individual gene can, for example, change an insect's leg into an antenna or exchange entire thoracic segments [83]. At the same time, it can take deletions of entire gene clusters to cause the developmental programs associated with different types of appendages to interfere with each other [77]. In this way the developmental program biases mutations to have coherent and interesting effects, thus pushing random search in promising directions. So not only is DNA tightly coupled with morphogenesis; both are tightly coupled with evolution.

Finally, this developmental program, and its parametrization by DNA, is universal among bilateral animals. The genes that direct the formation of our basic body plans — known as Hox genes — evolved over 550 million years ago and are shared by 99% of all known animal species [30]. So the diversity of the animal kingdom demonstrates the huge variety of forms that this single basic developmental program can generate, and the conservation of these genes demonstrate how valuable it is in evolutionary terms.

On the whole, biological morphogenesis utilizes a compressed search space (the genome) tightly coupled with a universal, distributed, and discrete fabrication process (developmental programs) to guide a powerful search strategy (mutation and recombination with natural selection). It serves as the most prominent example of what optimization over physical processes can achieve.

1.4 Simulation, Representation, Optimization, and Evaluation

Throughout this thesis I will discuss inverse methods in the context of four foundational layers: simulation, representation, optimization, and evaluation. Within

each layer, I seek to achieve greater flexibility by challenging common assumptions in existing practice. Between the layers, I seek tighter integration as inspired by biology.

The **simulation** layer models the behavior of a design. In general, the simulation can model any branch of physics using any techniques; the only essential requirements are that it can simulate any design in the design space, and that it provides enough fidelity for an accurate objective function to be defined. This thesis focuses on particle systems, which are an adaptable, mesh free family of simulation techniques well suited to the discontinuities and messy phenomena that arise in inverse problems. I have developed a particular type of particle system, and implemented it for three different compute architectures: CPU, GPU, and a custom system developed at CBA called DICE. Other implementations are ongoing.

The **representation** layer defines the space of possible designs. For many applications a design consists purely of geometry, but it may also include control laws or other non-geometric features, up to and including the laws of the simulation itself. To facilitate representation independence, I have developed a design abstraction layer which can wrap arbitrary design spaces to make them compatible with my inverse problem toolkit. Design languages built in this framework will feature in both applications, particularly in the form of functional representations (FReps) of geometry.

The **optimization** layer searches for the design in the design space that minimizes the objective function. In theory any optimization method may be used, such as the simplex method, gradient descent, or evolutionary algorithms. I will discuss the relevant concerns for selecting an optimization algorithm for simulation based inverse problems, and provide an overview of the particular algorithm I rely on for my applications: the covariance matrix adaptation evolutionary strategy (CMA-ES).

The **evaluation** layer defines the objective function for the inverse design problem. It may take as inputs any and all features of the design, as well as any accessible internal state of the simulation. For the purposes of this work, I assume it produces a single scalar output, but multi-objective optimization techniques could be applied as well. The evaluation layer defines a specific inverse problem as much as it helps solve it, so it is difficult to provide general recommendations for its construction. So

my discussion of this layer primarily revolves around commonalities drawn from the applications, and for this reason it appears after the applications are presented.

1.5 Contributions

This thesis contributes a set of tools for inverse methods over particle systems, and demonstrates their use with two applications.

In chapter 2, I present a portable and generic particle simulation. It achieves state of the art performance on GPUs, but can also be run on CPUs or DICE, a new type of modular computing system being developed at CBA. A Verilog implementation is being developed, which will enable it to run on FPGAs, and will drive the specifications for a custom superconducting integrated circuit for particle simulation.

In chapter 3, I present a framework for constructing custom design languages. It is conceptually defined by static single assignment statements that conform to a context-free grammar, and implemented as a DAG with several different node types. All design languages constructed with this system are inherently parametric, and can be easily integrated with optimization routines.

Chapter 4 discusses optimization algorithms that can be used in tandem with the simulation and representation tools.

The first application is a design study for gear tooth profiles, discussed in chapter 5. I demonstrate gradient free optimization of a dynamic and relatively stiff rigid body simulation.

The second application (chapter 6) optimizes a course grained force law, in order to allow simulated plastic coupons on a virtual Instron to produce stress strain curves that match measured data. This demonstrates the automatic construction of macro-scale course grained models, as well as material memory arising from particle distributions rather than explicit memory kernels.

Chapter 7 discusses some conclusions on inverse problem definitions based on the two applications.

Taken together, I hope that these tools and results can contribute a foundation

for further research into design and simulation with inverse methods over particle systems.

Chapter 2

Simulation

Simulation is the foundation of the work discussed in this thesis. In this chapter, I first discuss some of the shortcomings of current practice, and how they can be addressed with particle systems. This is followed by a primer on the theory and practice of particle systems, and discussion of four specific implementations I have written or am working on.

2.1 Background

2.1.1 Motivation

Simulation is now ubiquitous in both academia and industry. But it also a common paint point for engineering workflows.

One significant issue is the disconnect between the representations of geometry used for design and simulation. All the most popular CAD packages utilize boundary representations, usually in the form of analytic surfaces but also sometimes as surface meshes[1]. Many of the most popular simulation packages, however, rely on the finite element method (FEM), and as such utilize volumetric meshes. The process of meshing a CAD model for simulation is computationally expensive, and one that requires substantial manual oversight. The results of FEM simulations can depend sensitively

[1]There are alternative, volumentric representations that have a number of advantages for many domains, but they have not yet seen widespread adoption [46].

on the quality of the mesh [70], so it is not a process that can be overlooked.

The development of new simulation codes is also often a painstaking process. While there are only four known fundamental physical forces, there is a bewildering variety of simulation techniques and physical approximations. Particularly challenging is the development of multi-physics models, which involve coupled interactions between different domains of physics. Similarly, multi-scale phenomena are also particularly challenging, i.e. those in space or time scales of various pieces of the simulation vary by several orders of magnitude.

Both these challenges are recognized in industry and research. The Defense Advanced Research Projects Agency (DARPA), for example, has in recent years funded research addressing these issues. The Fundamental Design project aims to develop new, physics aware building blocks for encoding designs at the conceptual level [25]. The Computable Models project aims to reduce the time required to develop novel multi-physics, multi-scale simulations [24]. CBA is a performer in both projects, so these projects have both funded and guided my research.

2.1.2 What Counts as Reality?

There is a philosophical point underlying much of this chapter that it is best to address head on. At the most fundamental level, all of modern physics is described in terms of interacting particles. Bodies with mass interact with each other via the gravitational field described by general relativity, and the standard model describes a small zoo of particle types (fermions) whose interactions are mediated by yet more particles (bosons). Our best understanding is that all other domains of physics are at root patterns that emerge from the interactions of large numbers of these basic particles and forces.

But it's not practical to appeal to these particles to explain these patterns, whether via analytic calculation or numeric computation, except for the simplest systems. So physicists have developed a huge variety of other mostly independent theories that approximate the net effects of these particle interactions for particular scenarios. Many of these theories are expressed as partial differential equations (PDEs), which

approximate the interactions of many particles as a continuum. This representation is mathematically convenient, and often elegant. For many phenomena, such as heat transfer in a homogeneous solid, or laminar flow of a fluid, PDEs can make predictions accurate enough for almost any practical purpose.

But PDEs are unambiguously an approximation of the underlying particle reality, which becomes all too apparent when one looks at more complicated phenomena. In turbulent flows, for instance, the PDEs (i.e. Navier-Stokes) describe energy cascades in the form of vortex chains from large length scales down to small ones. This process can start at the kilometer scale (e.g. a hurricane) and only ends when the energy dissipation rate and the viscosity become balanced, often at the millimeter scale. Physically, this is the length scale at which the coherent movement of the fluid breaks down into the incoherent thermal motion of the molecules that constitute it. So the molecular scale properties of a fluid end up being extremely important for understanding its macro-scale behavior. While it is an open problem — and one with a million dollar reward [17] — some prominent mathematicians are convinced that the PDEs themselves admit nonphysical solutions that blow up to infinite energy in finite time [90].

Additionally, except for the most basic cases, PDEs cannot be directly solved by computers. So when we implement simulations, we approximate yet again. The irony is that the PDEs, which are a continuous approximation of a discrete reality, are then discretized again for numeric solution. So most simulation codes are a discrete approximation of a continuous approximation of a discrete foundational theory. A fundamental thread throughout this thesis is that we don't need to go through PDEs along the way. Particle systems that interact in ways other than the fundamental forces of the standard model and gravity can approximate the underlying physics just as well as PDEs. This isn't to discourage the use of PDEs, or deny that they are an enormously powerful tool for proving properties of physical theories, just to point out that physical simulations not derived from them are not inherently invalid.

2.2 Particle Systems

2.2.1 History

The fundamentals of particle based simulation can be traced to Newton's *Principia*. In this iconic work he laid out both his laws of motion and the law of universal gravitation, and demonstrated that these laws produce the elliptical orbits of the planets and the parabolic trajectories of comets [71]. These laws still form the complete mathematical basis of (non-relativistic) n-body simulations today. Even more surprising is that in deriving Kepler's second law, Newton employed a geometric method that is mathematically identical to the integration scheme that will be used throughout this thesis (Verlet integration, see section 2.2.5) [36].

The first published, particle based simulation performed on a digital computer was Alder and Wainwright's 1959 study of equilibrium and non-equilibrium statistical mechanics based on elastic collisions of rigid spheres [4]. This study arguably founded the field of (computational) molecular dynamics. In quick succession more physically realistic studies were performed: Gibson et al's 1960 study of radiation damage in copper [33] and Rahman's 1964 study of liquid argon at the atomic level [75].

They soon found use for a wide range of other simulation problems, particularly those with messy physics and boundary conditions that are challenging for mesh based methods and finite difference methods. The geophysics community was a particularly quick adopter, particularly for the study of granular flows [22]. They also became common in vortex methods for computational fluid dynamics [18].

2.2.2 Modern Practice

Today there are a range of related simulation techniques, and the terminology can be opaque. "Discrete element simulation" is used as a somewhat general term, often encompassing molecular dynamics, discontinuous deformation analysis, and slight variations on the theme (distinct element method, generalized discrete element method, etc.). For the purposes of this thesis I take a wider view: I consider all of the following

methods to be particle systems, based on the observation that they can be integrated into a generic particle simulation in a relatively straightforward way.

Smoothed particle hydrodynamics (SPH) is explicitly a particle system. Rather than compute forces as pairwise interactions between particles, it convolves the particles with a weighting function to produce a field that then guides particle state changes [66]. SPH was originally developed for astrophysics problems, but is also increasingly popular for fluid-structure interactions, multiphase flows, and free surface flow.

The material point method (MPM) refers to its discrete elements as "material points", and is a standard particle system in every way except that it extrapolates (or interpolates) particle properties to a background grid in order to solve for accelerations and gradient terms (such as the deformation gradient) [89]. It is an increasingly popular tool for rigid and soft body mechanics. There is also a differentiable implementation that has been used for a variety of inverse design problems [41].

Position based dynamics is a purely particle based method for rigid and deformable bodies. It deals with particle positions as the most fundamental property, rather than forces (as described in section 2.2.4). This is achieved by using constraints to enforce relationships between particle positions, and using a modified form of integration [69].

Lattice gas automata (LGA) are also based on particles, but quantize the particle positions and momenta in addition to the mass parcels (i.e. particles themselves) [29]. The earliest models restricted particles to travel in the four cardinal directions at a constant speed, but these gases exhibit highly anisotropic behavior. Hexagonal grids mostly solve this problem, that is, having six momenta directions but still a single magnitude. Particle interactions are mediated via a collision rule, which is applied when multiple particles move into the same lattice node in a given time step.

A more modern extension of LGA is the Lattice Boltzmann Method (LBM) [15]. It has largely superseded research and practical interest in LGA. LBM is an extension of LGA in which the lattice nodes house particle distributions rather than individual particles. The discrete collision rule of LGA is thus replaced with a continuous collision operator, which tends to return the particle distributions to their equilibrium

state. This relaxation is typically modeled with the Boltzmann equation, hence the name.

Finally, peridynamics is a new formulation of classical continuum mechanics designed to handle structural discontinuities (such as cracks) where the classical theory breaks down [86]. The theory is still decidedly a continuum one, unlike the other particle methods discussed here. However, it is possible to use a mesh free discretization scheme for numeric solutions that naturally fits within discrete element frameworks [87].

2.2.3 Comparison with FEM

Particle systems have many advantages compared to the finite element method (FEM). Most of these ultimately stem from the fact that particles are mesh free. As mentioned earlier, FEM simulations can depend sensitively on the quality of the mesh [70], and meshing itself can be a computationally expensive operation that requires substantial manual oversight. Conversely, to prepare a particle based simulation one need only seed particles within the simulated material. These can be placed on a regular lattice, or in a random fashion. Both methods are simple to compute quickly in a highly parallel fashion, and can be automated reliably. So the absence of a mesh in particle simulations replaces an error prone, expensive preprocessing step with a simple, fast one. Additionally, in simulations with large deformations, it can be necessary to re-mesh repeatedly. No such step is needed for particle systems.

FEM simulations can also struggle with convergence. This is particularly true for simulations that develop regions of high stress (or other field variable) gradients — commonly called "shocks" — and those with topological changes in the meshed structures. Particle systems handle these situations much more gracefully. Topological changes present no problems, and large gradients are fine as long as the stability conditions (such as the Courant–Friedrichs–Lewy condition) remain satisfied. In fact, some adaptations of FEM designed to handle large gradients and topological changes essentially end up incorporating particles in one form or another, such as moving nodal points [65].

2.2.4 Fundamentals

The basic algorithm is simple to describe. A simulation consists of a set of particles and a force law. Each particle has (at least) a mass, position, and velocity. The force law describes how the particles interact. In the simplest and still quite common case, this entails computing an interaction force for each pair of particles, as a function of their positions and velocities. More advanced formulations may add additional state to each particle, such as temperature, angular velocity, etc. Others compute a field based on the positions and other properties of all particles which is then used to determine the pairwise interaction forces.

Given the interaction forces, Newton's second law then specifies a system of ordinary differential equations that can be integrated to determine the particle trajectories. For instance, assume a three dimensional simulation of n particles. Let each particle have mass m, and let the positions of each particle over time be given by $\vec{x}(t)$ (a function from $\mathbb{R}_{\geq 0}$ to $\mathbb{R}^{3n}$). If the force between particles depends only on their positions, as is the case for a conservative force, then Newton's laws indicate that

$$m\ddot{\vec{x}} = \vec{f}(\vec{x})$$

This differential equation can be numerically solved with a variety of techniques.

2.2.5 Integration

In this thesis, I use the Verlet-Störmer method, otherwise known as leapfrog integration [92]. This is a second order integration scheme (in time), that requires comparable computational effort to the simple but only first order Euler integrator. Additionally, it is a symplectic integrator, meaning it preserves the geometric properties of phase space. This enables it to preserve the energy of conservative system even better than higher order methods [36].

For the macro-scale problems considered in this thesis, it is desirable to add some amount of dissipation, as otherwise excited vibrational modes would persist indefinitely — clearly an unphysical behavior for commonplace macroscale materials. While

the standard derivation of the velocity Verlet method assumes that force is not a function of velocity, it is possible to adapt it to accommodate such force laws. This breaks the symplectic properties of the standard velocity Verlet algorithm, but since dissipative systems are not conservative this is not a problem in practice [34].

Altogether the basic algorithm I use is as follows. Let there be n particles. At some time step i, suppose we know the particles positions $\vec{x}_i$ and velocities $\vec{v}_i$ (both are vectors in $\mathbb{R}^{3n}$). Suppose also that we have a function $f(\vec{x}, \vec{v})$ that returns the accelerations on all particles at a given time, as a function of their positions and velocities. Internally, this usually involves computing the forces acting on each particle, then applying Newton's second law (i.e. dividing by the particle masses). Then the positions and velocities at the next time step ($\vec{x}_{i+1}$ $\vec{v}_{i+1}$) are computed based on the following steps.

1. Compute the relevant accelerations.

$$\vec{a}_i = f(\vec{x}_i, \vec{v}_i)$$

2. Take a half step for the velocities.

$$\vec{v}_{i+1/2} = \vec{v}_i + \frac{\Delta t}{2}\vec{a}_i$$

3. Take a full step for the positions.

$$\vec{x}_{i+1} = \vec{x}_i + (\Delta t)\vec{v}_{i+1/2}$$

4. Take another half step for the velocities.

$$\vec{v}_{i+1} = \vec{v}_{i+1/2} + \frac{\Delta t}{2}\vec{a}_i$$

This process can be repeated iteratively to integrate forward as far as desired.

2.2.6 Locality

Steps 2 through 4 as described above are completely generic. It is the calculation of forces in step 1 that distinguishes one particle system from another. Force laws vary widely between systems, so there isn't too much that can be said about them in general. But there is one essential commonality: locality.

Our best understanding of modern physics indicates that all macro-scale phenomena are governed by local interactions[2]. It is true that the law of universal gravitation, as formulated by Newton in 1687 [71], and Coulomb's Law, as published in 1785 [19], both involve "spooky action at a distance." But this is not the case for the modern, field based theories (i.e. general relativity and Maxwell's Equations) that later superseded them. And even when it is more convenient to use the earlier formulations, as is often the case with particle systems, the forces fall off rapidly with distance and so can sometimes be ignored past a certain point without adversely affecting fidelity.

Simulation of such systems can be greatly accelerated by exploiting this property. The naive algorithm for computing all forces in a particle system checks every possible pair of particles. For n particles, this means there are n^2 interactions to consider. But knowing that no two particles can interact beyond some distance d greatly simplifies this, since we only need to iterate over pairs of nearby particles. This can be performed in $\mathcal{O}(n \log n)$ time using tree based spatial data structures, or $\mathcal{O}(n)$ time by using a single pass bucket sort to place particles into geographic bins with side length d. The latter algorithm is used for all simulations discussed in this thesis.

Even assuming the use of an accelerated algorithm, the size of the interaction cutoff distance, d, has profound implications for performance. Suppose that particles are approximately evenly distributed in space (as is usually the case in gas systems at equilibrium); say m particles per unit volume. Then the sphere of radius d surrounding each particle will contain approximately $m\frac{4}{3}\pi d^3$ particles. This means that the density of particle interactions is $m^2\frac{2}{3}\pi d^3$ interactions per unit volume (the factor of one half is introduced since each interaction involves two particles). So doubling d

[2]Quantum mechanics is usually interpreted as a nonlocal theory, though this is negotiable if one is willing to cede the principle of counterfactual definiteness.

results in eight times the computational burden.

Though not employed in this thesis, it is also worth noting that locality can be of use even when the force laws cannot be truncated. The Barnes-Hut algorithm uses a quad tree to find an approximate solution to n-body problems with long range force laws in $\mathcal{O}(n \log n)$ time [8]. More difficult to implement, but ultimately offering better performance, is the $\mathcal{O}(n)$ fast multipole method [78].

2.2.7 Parallelization

To make the most of modern compute hardware, it is necessary to parallelize the simulation code. There are a number of strategies that one can employ. The capabilities of the target hardware (CPU, GPU, etc.) can have a big influence on their performance. These concerns will be addressed in the discussion of my particle system implementations for the CPU, GPU, and custom hardware, but it is still useful to introduce the general strategies here. In nearly all practical simulations, computation of the interaction forces dominates the integration time, so this step will drive the discussion throughout.

The most straightforward approach is to parallelize the iteration over particle pairs. This method is simple to implement, but scales poorly for large number of processors. This is a simple application of Amdahl's law [76]: even if force computation initially accounts for 95% of the compute effort, the program can only be made 20 times faster by reducing the time required for this step. And even assuming perfect scaling (so parallelization across n threads reduces the computational time by a factor of $1/n$), nearly 80% of this maximum speedup is achieved with 64 cores, a tiny fraction of the parallelism available in modern HPC systems. In practice, the diminishing returns of additional cores can become actively detrimental, sometimes well before this limit, due to the overhead of spawning threads (or other parallelization primitives), and sharing data between them.

An alternative strategy bases the parallelization on locality. This strategy effectively subdivides one simulation into many smaller simulations, which run independently except for some additional bookkeeping at their seams. In particular, each

sub-simulation is responsible for managing a particular region of space, including storing the state of particles within it, computing their interaction forces, and integrating their trajectories. Forces may act across region boundaries, however, and particles may move between them. Thus this parallelization scheme requires some additional overhead (relative to running truly independent sub-simulations) to deal with these cases.

The cost of this overhead varies with the size of the regions, the particulars of the force law, and the distribution of the particles (which is determined by the force law and the initial conditions). Size wise, the memory required to store particle data and the computational effort required to compute interaction forces are both proportional to the volume of the region. The data that must be transferred between neighbors, however, is proportional to the surface area of the region. Thus the overhead of communication between regions increases as the size of the regions is made smaller. So if one wants to use additional cores to run simulations over larger and larger volumes, this parallelization method scales perfectly. However if one wants to use additional cores to accelerate a particular simulation with a fixed size, only so many regions can be added before the overhead becomes too costly. While data transferred across seams is always proportional to the surface area of the regions, the prefactor is determined by the force law cutoff. And if the distribution of particles is sufficiently inhomogeneous, the particulars of this distribution will matter more than the averages discussed here.

This brings up an important point: different regions may end up doing different amounts of work. And since each region cannot perform its integration step until it knows all the forces that act across its boundaries, it must wait for its neighbors. This can lead to large amounts of compute resources being underutilized.

Even if all regions do have the same number of particles and interactions, random fluctuations in compute time can have a noticeable impact on performance. We can estimate the magnitude of this effect with a simple statistical model. Assume that compute times are normally distributed, but that each node can't begin computing frame $i+1$ until all its neighbors (and itself) have computed frame i. So the time t_n^i

at which node n advances to frame i is equal to $\max_m(t_m^i) + \max(N(\mu, \sigma), 0)$. Here the first max is computed over the node and its neighbors, and the second max simply prevents time travel.

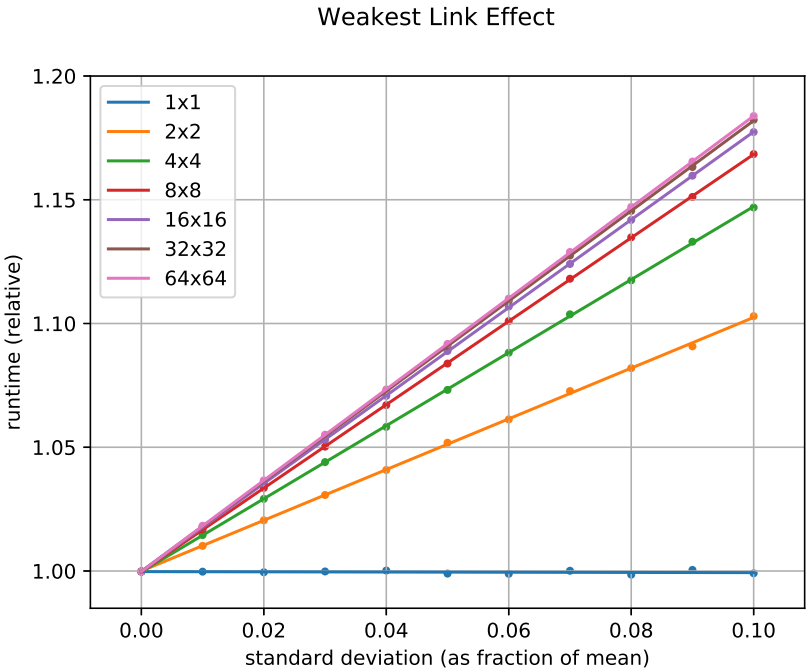

Figure 2-1: Weakest link effect for geographic parallelization schemes.

The strength of this "weakest link" effect depends on the number of nodes, and the compute time distribution. In particular, the relative slowdown depends linearly on the standard deviation of compute time, and roughly logarithmically on the number of nodes per edge of the array (so roughly log log on the number of nodes). Empirically, the total slowdown from the weakest link effect saturates at around 20%, assuming the standard deviation of the compute time is bounded at 10% of the mean compute time, regardless of the size of the array. This is good because it means that geographic parallelization schemes can scale arbitrarily without the weakest link slowdown following suit.

An important consideration for geographic parallelization schemes is how to divide inter-region interaction forces among regions. One strategy is to establish a set of priorities for each direction. For example, particles that interact across a boundary along the east/west axis could always be computed by the eastern region. This family

of strategies works well when the cutoff distance is small relative to the region size. When the cutoff distance is larger than the region size, it can be beneficial to use more complex schemes such as the neutral territory method [84].

2.2.8 Stochasticity

Mathematically, all the algorithms described in this chapter are deterministic. So one would expect simulations to be exactly reproducible. However, this is not the case for any of the implementations discussed below. This stochasticity arises from parallelization. When multiple computations are performed in parallel, then combined, the overall order of the operations involved is not guaranteed. Floating point arithmetic is not strictly commutative, so the sum of two floating point values $a+b$ is not always equal to the sum $b+a$. These tiny differences can lead to divergent macroscale behavior surprisingly quickly. Some reasons for this phenomenon are discussed in section 4.1.

2.3 Implementations

I have written three different particle systems implementations, targeting CPUs, GPUs, and a custom type of modular compute hardware developed at CBA known as discrete integrated circuit electronics (DICE). I am working on a fourth implementation in Verilog, that will be used both for running simulations on FPGAs, and as the basis for an application specific integrated circuit (ASIC). All of these implementations are based on the techniques discussed earlier in this chapter. All three of the complete implementations have similar organization and interfaces, and share some common software components.

2.3.1 Constraints and Measurements

Many simulated systems are not closed; energy can be added or removed by external forces. Such a capability is essential for the applications discussed in chapters 5 and

6. In all implementations, I model external forces via *constraints*.

Fundamentally, a constraint consists of a subset of particles within a simulation, and a rule for applying force to these particles. These forces are added to the forces regularly computed due to the simulation's primary interaction law (they do not replace them). For the applications discussed in this thesis, two different constraints are required: linear and radial. Linear constraints act to keep their particles constrained to a single axis of displacement, and radial constraints act to keep their particles a constant distance away from some specified center.

A linear constraint functions by canceling forces that aren't aligned with its axis. So a linear constraint set to keep a particle on the x axis, for instance, would apply forces along the y axis to counterbalance those arising elsewhere, while not influencing forces along the x axis. A radial constraint functions similarly, modifying radial forces and leaving tangential forces unaltered. However radial constraints don't nullify radial forces; they apply a net radial force of the correct magnitude to counter centripetal forces.

Note that this implementation differs from simply picking the particles up and moving them to the desired locations. Specifically, it alters forces, not velocities or positions. This has a number of advantages. First, it is easy to compute the amount of work performed by the constraints. This makes it easy to track the energy that is added to or removed from the simulation. Second, it avoids a variety of nonphysical failure modes. For example, if a particle is constrained to the x access via direct manipulation of its position, the y component of its velocity ceases to be meaningful. The regular integration procedure may lead to this component growing without bound, which will wreak havoc if the interaction law depends on velocity (as is commonly the case for dissipative systems). This can be fixed through appropriate manipulation of the particle velocities, which introduces similar decouplings between force and velocity, which can then only be fixed via appropriate manipulation of forces. So in my experience it is much better to start at the bottom and avoid nonphysical decoupling as much as possible. (Anyone reading my code may protest that I do in certain circumstances manipulate velocities and positions directly. These are legacy

features that I am working to remove.)

Constraints can also be driven, enabling control over the motion of particles along their permitted degrees of freedom. Specifically, particles constrained to motion along a certain linear axis can be driven to follow a certain trajectory along this axis. Similarly, the angular position of particles in a radial constraint can also be guided. Generic trajectories are easy to specify by writing new code (and thus recompiling), but my implementations also support a variety of paths that can be configured at runtime. Currently these include sinusoids, linear motion, step functions (useful for testing impulse response), and smoothed step functions. Motion along trajectories is guided by a proportional-derivative (PD) controller. As before, this controller sets forces rather than velocities or positions.

Finally, constraints also provide a useful abstraction for measuring properties of the system. Currently my linear constraints are capable of recording the average displacement of their particles, as well as the net force applied for driven trajectories. Radial constraints record their particles' average angular position or velocity. These are the measurements I have found useful so far, but other reductions over the particles within constraints would be simple to implement.

2.3.2 CPU

I began development of the CPU implementation in February 2019. It started as a serial implementation, was updated to include force parallelization, and ultimately was rewritten to support geographic parallelism. Figure 2-2 illustrates the performance of the simulation in its various incarnations, as measured by total particle updates computed per second.

Single threaded performance of this implementation is competitive with that of LAMMPS, a leading state of the art framework for molecular dynamics [53]. Multithreaded performance was never optimized as heavily, as my development focus for parallel implementations shifted to the GPU. Though the CPU implementation served as the template for all subsequent implementations, it is currently only maintained as a testbed for algorithmic experimentation since it has been surpassed by its

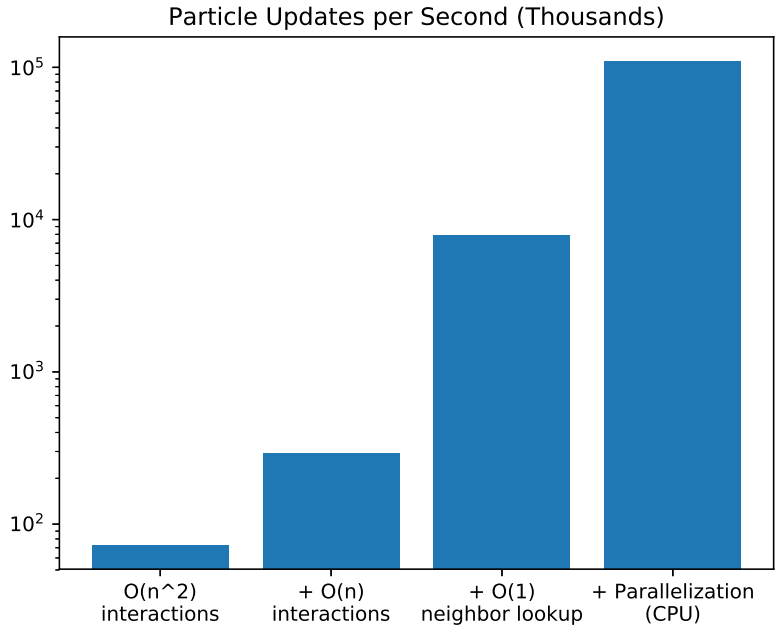

Figure 2-2: Performance of the CPU particle system implementation at algorithmic milestones.

successors in performance and relevance for my research interests.

2.3.3 GPU

Work on the GPU implementation began at the end of April 2020. Unlike the CPU implementation, it does not use geographic parallelization. However it is far more parallel than the force parallelism described in section 2.2.7, in that not only the force computations, but the integration and constraint calculations are also parallelized. This scheme works well for the highly parallel nature of the GPU.

Indeed, the GPU implementation can achieve more than one billion particle updates per second. This is an order of magnitude faster than the maximum performance I achieved with the CPU implementation. It also compares favorably with Flex, Nvidia's in house particle system implementation, which uses a position based dynamics model. In particular, across a variety of use cases, the authors of Flex report performance between 2 million and 20 million particle updates per second (see Table 1 in [62]). However these numbers are from an older model of GPU (the GTX680),

Table 2.1: Time steps computed per second using the author's CPU implementation and LAMMPS.

Number of particles	**Author's impl. (1 core)**	**LAMMPS (1 core)**	**LAMMPS (28 cores)**
3,438	6,326	3,849	24,102
7,776	2,858	1,692	9,500
14,078	1,591	913	7,034
56,958	411	190	1,407
358,393	67	22	353

which performs 16 times slower in the Geekbench 5 benchmark than the V100 that I used [21]. Adjusting for this, Flex would achieve between 32 and 320 million particle updates per second. This is still fewer particle updates per second than I see with my simulation, but because Flex is constraint based rather than force based it is unconditionally convergent and thus can run with larger time steps. So a direct test of the same physics problem would be needed for a truly apples to apples comparison.

Because of its performance, the GPU implementation is the one I use for all current design and simulation studies.

2.3.4 DICE

The discrete integrated circuit electronics (DICE) implementation was conceived as a demonstration application of the DICE project. DICE is a system that enables three-dimensional assembly of high performance computing systems from basic computational building blocks. This architecture unifies the conventional design of HPC packages, boards, blades, and systems in a single direct-write process. It is being developed at CBA with primary funding from a DARPA seedling grant [27].

The current generation of DICE hardware uses Microchip SAMD51J20 ARM Cortex M4F microcontrollers. The smallest versions use 3.5 x 3.6 mm wafer-level chip-scale-packages (WLCSP), as pictured in figure 2-3, while other incarnations use larger packages. This version is assembled in a three-dimensional cubic lattice. Each node can communicate with its six neighbors. To exchange information with more distant

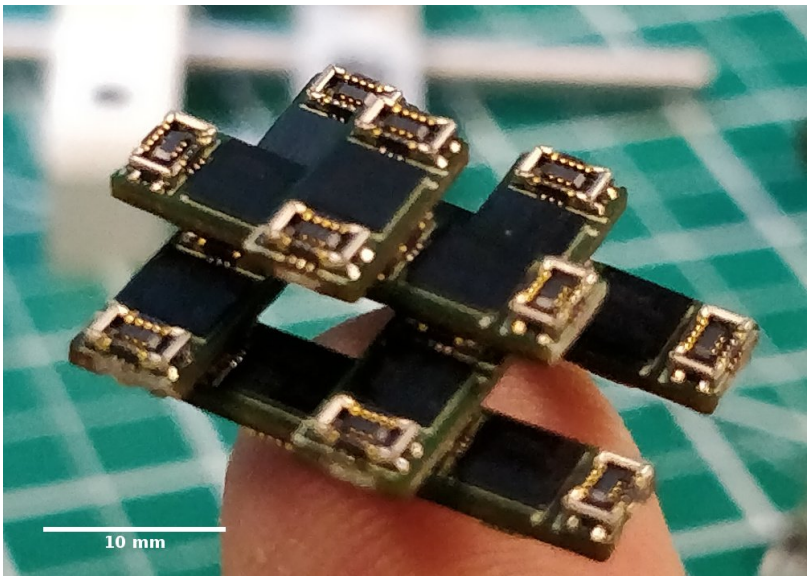

Figure 2-3: An assembly of eleven DICE nodes.

nodes, messages must be passed along a chain of intermediates. This locality is key to DICE, as it greatly reduces the complexity of interconnect found in current state of the art supercomputers.

My particle system implementation for DICE is currently restricted to two dimensions (as are the CPU and GPU versions). It uses a very literal version of geographic parallelism, in which each DICE node handles one region of virtual space, and the tiling of regions in virtual space mirrors that of the DICE nodes in physical space. To transmit data between neighbors, it must be serialized and transmitted by the SAMD51's UARTS. This makes the communication overhead discussed in section 2.2.7 a primary concern, as communication can easily become a bottleneck.

Implementing a particle system simulation required the development of several layers of supporting software which will be reused for other applications. Collectively these layers constitute an application programming interface that we call the DICE programming model, and define the DICE runtime environment. The first layer is a hardware abstraction layer (HAL) that operates the SAMD51's hardware peripherals. The next layer is a data link layer responsible for transferring raw byte strings between neighboring nodes, informed by the data link layer of the OSI model [26]. Finally, the third layer implements a token passing model which enables the preservation of

message semantics via a simple system of identifiers and types, and synchronization of nodes based on querying for the presence or absence of certain tokens. This token model provides an interface very similar to the Message Passing Interface (MPI) that is used in state of art supercomputers today [93]. However there is one crucial difference: all communication in the DICE token passing model is local, so each node can only send messages to its immediate neighbors.

While the DICE firmware token layer uses the DICE data link layer, it is easy to implement the same token interface on top of other backends. I have done this for thread level parallelism on CPUs (I simply use the C library's string copying functions to transfer bytes, in place of the SAMD51 UARTS). This enables DICE application code to be compiled for different systems simply by including the appropriate token model header file. This has proven invaluable for DICE software development, since debugging on a modern computer is far simpler than debugging a novel hardware/software system. It would be easy to write an implementation of the DICE token passing model that uses MPI as its backend, which would then allow DICE applications to be trivially ported to the current generation of supercomputers. Enabling portability to GPUs is trickier, as the parallelism model of GPUs differs significantly from CPUs, supercomputers, and DICE. Bridging this gap is a focus of ongoing research.

2.3.5 FPGA/ASIC

A development effort that is still underway is the implementation of a particle system in Verilog. Ultimately this is intended to create a custom chip dedicated to particle system simulation. In particular, CBA is collaborating with the Superconducting Electronics Group at Lincoln Laboratory, and we are targeting a Josephson junction based circuit that would become the first superconducting DICE node[3]. However it will also be interesting to run the Verilog version on large FPGAs, and compare

[3]The Superconducting Electronics Group's state of the art superconducting circuits have remarkable speed and energy efficiency, but it is not currently possible to place more than a few million Josephson junctions on a single die. DICE provides a path toward assembling a large computer out of many smaller and feasibly manufacturable pieces.

performance with CPU/GPU implementations.

Chapter 3

Representation

3.1 Background

To encode the most variation inside the fewest dimensions, it is desirable to use highly structured representations that are tightly coupled with their problem domains. This leads us to specialized representations, that don't generalize between problems. On the other hand, we don't want to trap ourselves with unnecessary assumptions, or to reinvent the wheel for every new problem. This leads us to unstructured, generic representations that are applicable if not tailored to a wide variety of problems.

Another key concern is the compatibility of the representations used for design and for simulation. Moving from the boundary representations used by most CAD programs to the volumetric mesh representations used for FEM simulations poses a significant challenge for more tightly integrating these two systems.

3.2 DAGCAD

My response to these questions is a framework for constructing design languages based on directed acyclic graphs (DAGs). Since I first applied it to CAD style problems of solid geometry description, I call it DAGCAD, though I now also use it for purposes that don't resemble CAD much at all. It's built from the ground up to support parametric design. It is informed by the design trees in common CAD packages, as

well as dataflow programming models. Note that it isn't a single design representation — it's a toolkit for quickly constructing parametric design languages, which can be as general or specific as desired.

Under the hood, it consists of a DAG with two different classes of nodes: design nodes and parameters. The design nodes describe primitive design elements and operations for composing them. For example, a language for constructive solid geometry (CSG) could define cubes, cylinders, and spheres as the basic atomic design elements, and operations such as unions, intersections, and differences as the compositional operations. The parameters represent numeric or other values that parametrize the geometry. So a sphere could have parameters describing its center and radius, a cube its position and side lengths, etc. Commonly these are simply used to store and reuse values, but they also support a simple language of arithmetic expressions. The edges in the DAG indicate the flow of information, so a compositional operation acting on two subdesigns would have those design nodes as dependencies, and a geometric operation parametrized by a given parameter would have that parameter as a dependency.

Formally, these DAGs can be defined as static single assignment (SSA) form statements in a context-free language. A series of node definitions is in SSA form if every node is defined exactly once, and each node's definition occurs after those of its dependencies. SSA forms are commonly used as intermediate representations in modern compilers [23]. Any DAG can be expressed in SSA form: one need only assign a unique identifier to each node, and order the node definitions based on a depth first traversal. Conversely, every series of node definitions in SSA form describes a DAG, since the presence of cycles or a lack of referential integrity would violate the definition of SSA.

But although an SSA form does fully capture the global structure of the DAG, it cannot capture the local structure. Specifically, we want to guarantee that (1) no parameter node depends on a design node, (2) every node has the correct number of dependencies. These local properties can be captured by a context-free grammar [16] that is applied to node definitions. Property (1) is required of all DAGCAD gram-

mars, while the details of property (2) are defined by each specific design language implementation. In my codebase, this is achieved by defining a "signature" for each type of design node.

In all the currently implemented design languages, all node types are valid inputs for all others. So for the sake of simplicity, this assumption is embedded in the framework as well. But future versions could relax this, and enable each type of node to require that its inputs be of certain types. In general, these sorts of constraints are called type systems, and they can become quite complex (specifically, computationally universal and thus undecidable) [12]. But for the limited use case described here, a context free grammar would be sufficient.

To summarize, at the conceptual level, languages in DAGCAD are described by SSA form statements that conform to a context-free grammar. Implementation-wise, this boils down to manipulating a DAG with certain invariants enforced on its edges. With this understanding, we can now discuss the benefits of such a representation.

Parameters with no dependencies represent the inputs of a design. These can be found automatically, and exposed as a single vector so that it's as easy as possible to interface DAGCAD designs with optimization tools. When desired, input parameters can be marked as constant, so that they are not surfaced for optimization purposes. Note this process of collecting free parameters depends only on the DAG structure; it is completely agnostic as to the semantics of the design nodes or the parameter themselves. This facilitates a separation akin to the genotype/phenotype distinction discussed in section 1.3.

This separation of concerns also enables a generic graphical model editing tool. Currently I only have a rough prototype, since I primarily use DAGCAD in code and haven't developed a strong desire for a better graphical tool. For portability, what I have written is implemented in JavaScript and runs in the browser. The intention is that it will communicate with the C++ backend via a web socket. The existing implementation is pictured in Figures 3-1 and 3-2.

The DAGCAD backend is implemented as a set of graph processing utilities in C++. The implementations differ from standard graph processing libraries in a few

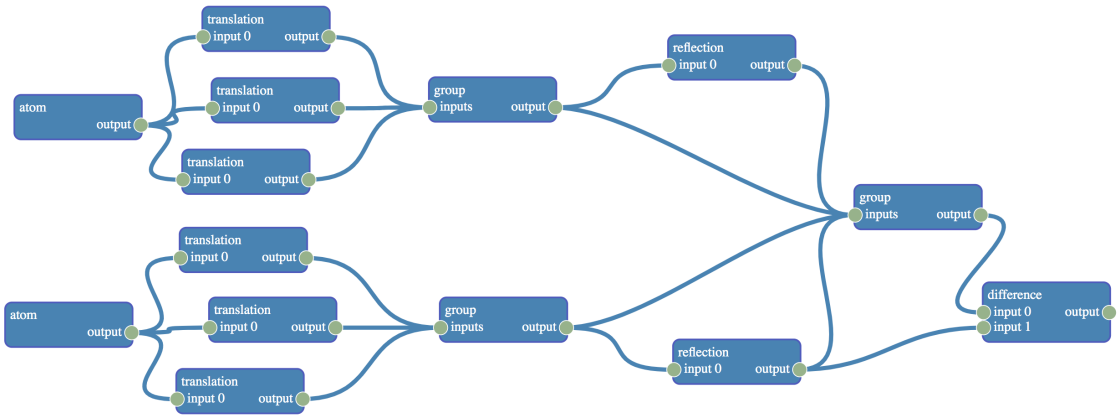

Figure 3-1: A portion of a model viewed with the prototype generic model editor. This design uses a specialized design representation for discrete robotics, modeled after work by Will Langford [54].

important ways. For one, my implementation is statically allocated. This makes it easy to use DAGCAD in bare bones embedded systems, such as the current generation of DICE (see section 2.3.4). Second, the rules for constructing edges are somewhat peculiar compared to most mathematically inspired implementation. Each node has an unordered list of outputs (so in mathematical terms, a set), but an ordered list of inputs. This enables non-commutative operations, such as geometric differences. Finally, my graph utilities are templated to make implementing new design languages as painless as possible[1], so the interface for extending these classes differs from more generic graph processing utilities.

3.3 FReps

I use functional representations (FReps) to describe most geometries used in this thesis. FReps encode geometry analytically, specifically as a scalar valued function interpretable as a distance field. Thus the surface of an FRep model $f(x, y, z)$ is the set of points for which $f(x, y, z) = 0$. The interior of a model is the set of points for which $f(x, y, z) < 0$, and exterior those points such that $f(x, y, z) > 0$ (some authors

[1] I am aware that many would rightfully consider "painless" and "C++ templates" to be incompatible. Future versions may provide a more accessible interface.

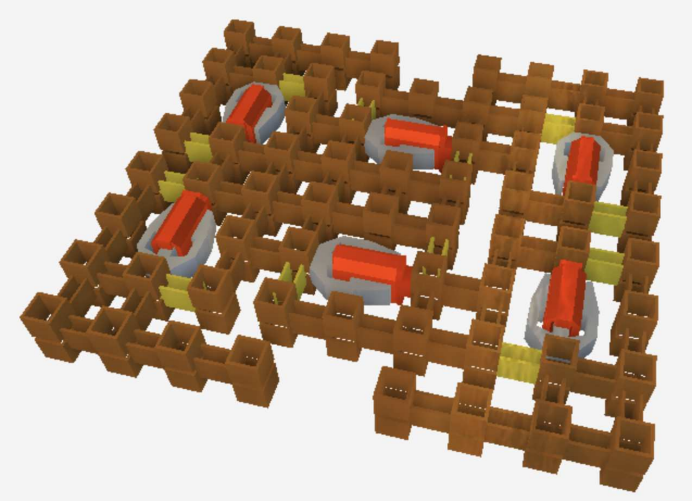

Figure 3-2: A rendering of the model of which a portion is depicted in Figure 3-1. This is a modular walker robot, as designed by Will Langford [54].

use the opposite convention for sign). FReps are descriptive enough to form large, generic design spaces, but they also enable easy parametrization of low dimensional, specialized models.

FReps are ultimately just mathematical expressions, so my DAGCAD FRep implementation reduces down to a DAG that only has parameter nodes. It would be reasonable to implement a set of design nodes for unions, intersections, etc., but I have chosen to keep the elegance of the pure parameter model, and use external helper functions to encode these operations.

Using external helper methods also enables extremely clean syntax for constructing FReps. By overloading the basic arithmetic operators in C++ for FRep nodes[2], it becomes possible to build FRep DAGs by writing expressions such as "x * x + y * y - 1.0" rather than explicitly building up nodes and edges. This functionality is inspired by libfive, a wonderful and fully featured FRep library written by Matt Keeter [45].

[2]Technically, I use a dedicated FRepBuilder class rather than the literal FRep DAGCAD nodes for memory management reasons.

3.4 Particles

Within the simulation layer, particles are the natural representation of geometry. Transforming from an FRep to a particle representation is far simpler than meshing a boundary representation: FReps tell you which points are in their interiors, so it's trivial to fill them with particles. Furthermore this process is embarrassingly parallel. Currently I perform this on the CPU. If it becomes a bottleneck, parallelization on the GPU would yield far greater performance. Since my DAG implementations are statically allocated and array based, they would be easy to port to the GPU. For even greater performance, a dedicated evaluation algorithm for FRep rendering on the GPU could be used [47].

The use of particles within the simulation raises the possibility of just using particles as the design representation as well. Then the design and simulation representations would be directly interchangeable. This is outside the scope of this thesis, but will be considered in future work.

Chapter 4

Optimization

With an appropriate representation and objective function, a wide range of optimization algorithms can be used. In this chapter I discuss some properties that make simulation based objectives particularly challenging, and review three classes of optimizers.

4.1 Chaos

Objective functions based on simulation can exhibit a variety of severely pathological behavior. This is ultimately all attributable to chaotic system dynamics, but the ways this can manifest are varied and can be quite surprising to those who have not studied them before. They have surprised me repeatedly.

Chaotic behavior is usually described as sensitive dependence on initial conditions. In other words, a very small change in initial position or velocity can quickly lead to very large changes in trajectory. This alone, however, is not sufficient to define chaos. Initially nearby particles falling due to a constant gravitational force, for example, will grow arbitrarily far apart, but their trajectories are not chaotic. Another necessary condition is topological mixing. This states that if you take any two regions in phase space (for our purposes, phase space is just the vector space formed by all the particles' positions and velocities) and integrate one forward according to the dynamics of the system, it will eventually intersect the other. More intuitively, chaos requires not just

that trajectories diverge, but that they get close as well. Any two molecules of water in a quickly stirred bucket will at some time be arbitrarily far apart (subject to the confines of the bucket), but at another they will be arbitrarily close.

This has important ramifications for us. If trajectories only got farther apart over time, then our job would be easy. Large regions of phase space at the end of a simulation would map backwards to specific small regions of phase space at the beginning. Every question of the form "How does *this* end up happening in the simulation?" would have a clear answer. But because phase space becomes completely scrambled over time, a neat region of phase space at the end of a simulation can map back to a fractal dust of initial conditions that defies any other description. When this is the case, there is no local structure for an optimization algorithm to exploit. Worse, there simply may not be a meaningful answer to give to our question, other than "this happens in a million different ways, each one unique and practically impossible to reproduce."

These facts can be difficult to reconcile with our intuition. Isn't the system totally deterministic? Yes, mathematically there's nothing random about it. (Let's ignore for a moment the implementation specific stochasticity discussed in section 2.2.8.) Can't trajectories in phase space never cross? Also true. If you integrated forward all points in phase space, each path would at times be arbitrarily close or arbitrarily far from any other, but none would ever cross. Since we're only integrating for a finite amount of time, can't we limit the amount of mixing? Again this is true, but it doesn't mean there won't be enough to thwart all feasible optimization attempts.

To close, let me reference two results from the literature that further demonstrate the peculiar power of chaos. A classic result from Cristopher Moore is that certain long term questions about the evolution of dynamic systems are unanswerable in principle [67]. He proves this by constructing a Turing machine inside a dynamical system, thus relating dynamical systems to the halting problem. Concretely, this means that for chaotic systems, there is no algorithm that can tell you if an arbitrary initial point in phase space ever reaches a certain other region[1]. Second, a more recent paper used

[1]This may seem to contradict the principle of topological mixing mentioned earlier. Recall that

an arbitrarily precise integrator for celestial dynamics to investigate the stability of three-body systems. They found that without knowing the initial conditions down to the Planck scale, about 5% of a certain class of orbiting triplets of black holes could not be reversibly simulated to the point of breakup (i.e. when one black hole is flung from the system) [11].

The conclusion we have to reach from all of the above is that you can't always win against chaos. If your simulation exhibits chaotic behavior, and this shows up in your objective function, there may be no optimization algorithm that can help you. That said, not every system we are interested in is chaotic. And even when chaos is present, it may be confined to certain avoidable regions of phase space, or it may be mild enough that it doesn't doom all optimization attempts. Sometimes an objective function can be rewritten to be less sensitive to chaotic noise, while still capturing the relevant features of a problem.

4.2 Gradient Methods

Optimization algorithms that make use of gradients typically converge faster than those that do not. As such, gradient based techniques are quite common in large scale practical optimization problems. Stochastic gradient descent, for instance, is nearly ubiquitous in training large deep neural networks. But such algorithms only work, of course, when derivatives are available.

Differentiating through a simulation can be tricky, but mathematically is completely feasible. MIT's computational fabrication group has released multiple recent papers demonstrating just this [41, 39]. In particular, they use these gradients to guide optimization over many different physical systems.

I am interested in differentiable simulation and gradient based optimization over it, but it is not within scope for this thesis. For all the reasons discussed in the previous section, for some physical systems gradients will inevitably be swamped by

chaos is not present everywhere or not at all; so trajectories may mix within a region of chaos, and either eventually break free or not.

chaotic noise, if not tend toward infinity. Even barring severe chaos, it can be difficult to compute gradients in a numerically stable way, further limiting their scope. In discussions with the authors of some of the papers cited above, the number of time steps that they can differentiate through is limited. So in the interest of generality, and allowing long time horizons within my objective functions, I will not currently use gradient based optimization methods.

4.3 Gradient Free Methods

Gradient free methods may not always converge as quickly, but they can be more robust to noise. In this thesis, I use an evolutionary algorithm called covariance matrix adaptation evolutionary strategy (CMA-ES). It iteratively refines a multivariate normal sampling population distribution to converge to a minimum in a coordinate independent manner. When the algorithm begins, the distribution is set to cover nearly the entire space of possible solutions. In each generation, samples are drawn from the distribution, and the mean and covariance matrix of the distribution are updated in an expectation-maximization step based on the best performing samples [37]. In this thesis I use the C reference implementation of CMA-ES. However in discussions with its author, he advises that the Python implementation is currently better maintained and more up to date.

To generate samples from the current distribution, it is necessary to find a matrix A such that $A^T A = C$ (where C is the current covariance matrix). If it is known that C is positive definite, the Cholesky decomposition can be used, but in the general case the algorithm relies on a spectral decomposition (the covariance matrix is always a real symmetric positive semi-definite matrix), thus incurring a large computational cost in high dimensions. For large enough N, even the storage required for the covariance matrix itself becomes prohibitive.

In design spaces with few enough dimensions, however, CMA-ES is a suitable algorithm for a variety of reasons. First, it meets my demand of being gradient free. Particle swarm optimization, another popular biologically inspired algorithm, is

not coordinate independent and can converge orders of magnitude more slowly than CMA-ES for certain rotations of the objective function [38]. Additionally, CMA-ES is designed to modify most of its hyper-parameters automatically, so it requires minimal tuning. Finally, it is in theory equally effective in spaces ranging from a few to a few hundred dimensions, and by adjusting the step size and population size the degree to which it performs global or local search can be tuned.

In practice, I have encountered moderately sized problems (i.e. around 20 dimensions) that CMA-ES struggles to optimize. I do not yet have enough information to say whether this is due to a pathological objective function, or due to the algorithm. In future work I plan to investigate modifications to CMA-ES that might address this problem, and allow it to function in higher dimensional spaces. First, one could compute an estimate of the covariance matrix by iteratively finding a fixed number of large (positive or negative) eigenvalues. This could be accomplished with the power method, or the more sophisticated Lanczos algorithm [94]. The resulting eigenvectors could then be used to generate samples from the estimated distribution directly. A more general — and potentially more powerful — approach would parametrize the distribution not as a multivariate Gaussian, but as a cluster weighted model of local basis functions [32].

Chapter 5

Application: Gear Design

As a first demonstration of this inverse design framework, I present the optimization of gear tooth profiles.

5.1 Methodology

5.1.1 Representation

Multiple representations are used, but there are some commonalities between them.

The gears are always represented as FReps[1]. All representations include a circular gear body. The inner portions of the gear bodies are constrained with radial constraints (see section 2.3.1). The left gear is driven in a sinusoidal fashion; the right gear is not driven but is used to measure the average angular displacement of the constrained particles. The gear bodies have hollow centers, since particles in the interior of the constraint add computational burden but do not affect gear interactions.

The primary representation for the inverse design studies is based on a union of circles. Each circle has three free parameters: the x and y coordinates of its center, and its radius. This is a generic representation, since with enough circles one could approximate any shape to arbitrary accuracy. I considered varying numbers of circles in different experiments, which will be mentioned in section 5.2. This representation

[1]Initial versions used libfive [45], later versions use the FRep library discussed in section 3.3. The choice of library makes no difference once the FReps are rendered to particles.

is trivial to implement as an FRep.

The use of the circles also varied by experiment. In some, the circles describe a single tooth, which is radially patterned on top of a gear body with a fixed size to create a complete gear. In others, the circles are not radially patterned, so each tooth must be independently constructed with a subset of the available circles (the gear body is still a constant). The former model enables high resolution gear teeth to be designed, while keeping the overall dimension of the representation space low. However it assumes symmetry, which means it includes part of the engineering knowledge that we would like to recover. The latter model requires a much higher dimensional representation space for comparable gear tooth resolution, but is more general.

In both of these models the distance between the gears is not fixed, but controlled with an additional parameter. So the total dimensionality of a model with n circles is $3n + 1$.

These generic representations were compared to an involute gear model, also encoded as an FRep. This model has three free parameters: addendum, dedendum, and pressure angle. It serves as a baseline for comparison.

5.1.2 Evaluation

The primary evaluation criteria is the integrated squared error of the angular position of the right gear (relative to its expected position) over a fixed amount of simulation time. An ideal set of gears would present negligible resistance when decoupled from external loads, so the deviation in the angular position of the left gear from its programmed trajectory should be minimal. Similarly, if the gears are meshed well, the right gear will always an have equal and opposite angular displacement, compared to the left gear. This establishes the expected angular position for the right gear.

As mentioned in section 5.1.1, the angular position of the left gear is driven sinusoidally. The amplitude of the driving signal is π radians, so that the gear oscillates between a half turn counterclockwise and a half turn clockwise, relative to its initial position. The simulation is run for one full period of the sinusoidal driving function, so overall the left gear rotates π radians counterclockwise, then 2π radians clockwise,

and finally π radians counterclockwise, returning it to its original position. For gear representations that don't enforce radial symmetry, it's critical that the gears complete a full revolution at some point so that all teeth are engaged. It's also critical that the gears change direction, as they are most likely to bind while doing so.

Though the driving pattern described above spends equal time traveling clockwise as counterclockwise, it's not totally symmetric. In particular, the initial acceleration is always in the same direction. This introduces a subtle bias in the objective function, and leads to asymmetric gear teeth (see section 7.2). To remedy this, I run two trials that are mirror images of each other: one where the left gear initially rotates counterclockwise, and one where it initially rotates clockwise.

Finally, an important consideration is recognizing when the positions of the gears is undefined. In particular, if the gears completely bind, the driving force can eventually exert so high a force that the simulation becomes unstable. When the gears explode, the average angular position of the constrained particles is meaningless. By reducing the step size, it could be possible to avoid many of these explosions. But properly functioning gears don't produce extreme internal stresses (relative to dysfunctional gears), so it's a feature not a bug to detect such situations early. Luckily this is easy to do, as gear explosions lead to large numbers of particles exiting the bounds of the simulation. In these cases a penalty is applied, to incentivize the optimizer to find stable gear geometries.

5.1.3 Simulation

For all representations, the simulation is seeded by filling the FRep with particles arranged in a hexagonal lattice. The particles interact with a simple linear force law. Any two particles that come within one lattice spacing of each other experience equal and opposite repulsive forces. Particles that are neighbors on the underlying lattice also experience equal and opposite attractive forces when they are between one and two lattice spacings of each other. This attractive force falls to zero abruptly beyond a distance of twice the lattice pitch, which is highly nonphysical. But this only occurs at the point of failure, so is not an issue for any reasonable gear geometry.

5.2 Results

Multiple FRep based design representations were compared. The lowest dimensional model is simply a parametric involute model. There are three exposed parameters: addendum, dedendum, and pressure angle. All gears in this design space mesh properly, but the optimization algorithm appears to have found that shorter, stouter teeth are stiffer and thus permit less deformation and backlash. The optimized result of the baseline involute model is visible in Figure 5-1.

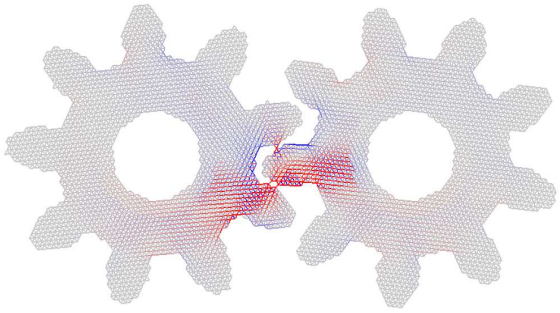

Figure 5-1: Optimized involute model.

The most successful generic model uses five circles to describe the tooth profile; the eightfold radial symmetry of the gear as a whole is baked into the model. Including the distance between the gears, this gives a total of 16 free parameters. Interestingly, the optimized design closely resembles a cycloidal gear. Unoptimized and optimized models are visible in Figure 5-2.

I also experimented with relaxing the radial symmetry constraint (i.e. allowing each tooth to evolve independently). Two such models were trained, one with six circles, and one with twelve. The six circle model converged to a highly symmetric solution as expected. Indeed its final form and score were very close to those of a model with the symmetry constraint but restricted to one circle per gear tooth. The twelve circle model, however, converged to an asymmetric and suboptimal local minimum, so further work is required.

Choice of model greatly impacts the convergence rate. This can be seen in Figure 5-4, which shows the convergence behavior of all four models discussed above. Recall that their numbers of free parameters are 3, 16, 18, and 36. Dimensionality is an

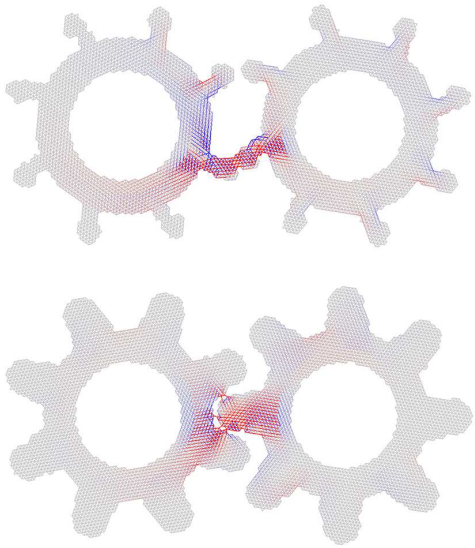

Figure 5-2: Optimization of gears described by a model that enforces eightfold radial symmetry. Above: an unoptimized design (best member of generation 0). Below: the optimized design.

important factor, but not the only factor.

Note that the lowest dimensional model (involute) converges faster than the second lowest dimensional model (symmetric circles), but it doesn't reach as low a score. This is noteworthy because it demonstrates that an optimized generic model can achieve a better result than a manually specified solution (at least within the not completely physically accurate bounds of this study).

5.3 Future Work

I have three primary goals for future work. The first is to remove more assumptions from the gear representation. Currently the model that does not enforce radial symmetry of the gear does not converge to a useful result. This may be solved simply by applying more computational power: distributed across multiple GPUs, far more designs could be evaluated, increasing the chances of finding symmetric designs. Since convergence is the primary issue, I could also look into alternative optimization algorithms.

I am also interested in modifying the objective function to motivate the replication

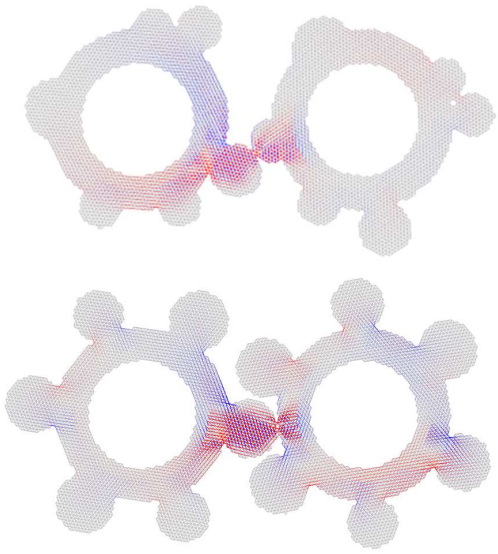

Figure 5-3: Optimization of gears described by a model that does not enforce radial symmetry. Above: a randomly generated sample from the design space (used as the center of the wide initial distribution). Below: the optimized design.

of gear types other than cycloidal. In particular, by incorporating a penalty for sliding gear contact, I hope to derive a geometry more akin to that of the involute family. This may be possible by adding a strong shear damping force, and penalizing designs that see a large drop in energy between the driven gear and the passive gear. A more direct approach would be to measure the shear damping force applied between non-neighboring particles (i.e. between the two gears) and penalize this quantity directly.

In an effort to speed up convergence, I would like to explore gradient based techniques. This would require making the simulation differentiable. Based on the issues discussed in section 4.1 it is likely that the gradients would not be stable over the number of time steps required for this problem. But the potential benefits are so great that I am eager to try.

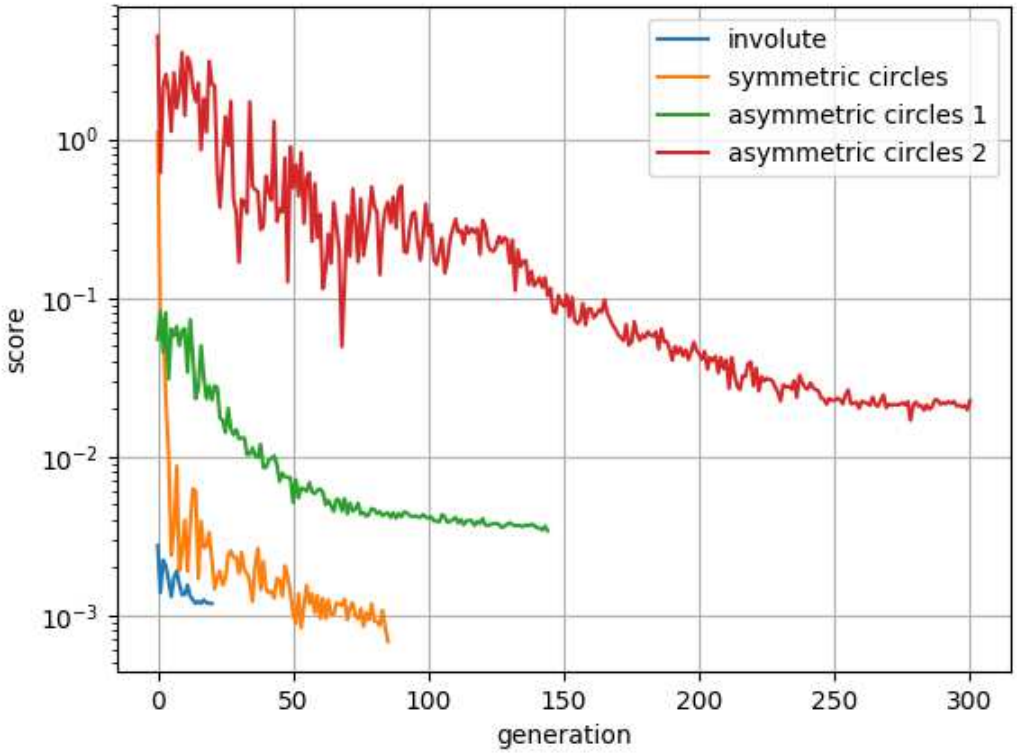

Figure 5-4: Convergence rates of the involute and circles models.

Chapter 6

Application: Force Law Search

6.1 Motivation

Simulation of phenomena that involve extreme geometry change, modest topological change, or any discontinuous geometric features are extremely tedious if not practically impossible to model with FEM. One such problem is plastic deformation. The large strains that can be exhibited before failure, and the discontinuous changes that occur at failure, pose a huge challenge compared to the comparatively trivial problem of elastic deformation. As we saw in chapter 2, particle methods are well suited to such challenges.

Unfortunately, it is still difficult to develop an appropriate set of interactions rules to simulate macro-scale plastic deformation with particle systems. At the atomic or molecular level, there is a relatively straightforward path from basic physics to interaction forces, i.e. based on van der Waals forces or density functional theory (DFT). But no current supercomputer comes close to being able to model even a centimeter scale plastic coupon at the molecular level in a reasonable amount of time, so this isn't of any use for macro-scale problems. As a result, it is common to use coarse grained models, whose interaction rules are designed to be valid for groups of atoms or molecules. These theories have been extremely successful in many domains [48]. There are still two significant drawbacks. First, most course grained models are developed for mesoscale simulations, not macro-scale. So they don't help us with

macro-scale problems. Second, even at the mesoscale they can be extremely tedious to derive.

The most common technique for deriving course grained models is the Mori-Zwanig formalism, which uses projection operators to split the dynamics of a simulation into relevant (i.e. faithfully simulated) and irrelevant portions (which are modeled as noise). This formalism tends to produce interaction laws that are no longer local in time, that is, they can have significant path dependence. This necessitates the introduction of memory kernels to model this longer time scale system evolution [59, 60].

There are alternatives to course grained models as well. The field of system identification is associated with the development of forward models for model predictive control. System identification problems typically use general statistical methods to fit black box models of plant behavior [61]. These models typically encode simple mathematical relationships between low dimensional inputs and outputs, making them not ideally suited to descriptions of internal stress and strain fields. Models that do capture a finer degree of physical detail tend to be specialized (e.g. [49]).

Constitutive models can have their parameters tuned to better match observed behavior. Often these models are based on PDEs, and thus are not suited to topological changes. When they are based on particle interactions, they typically assume domain specific parametric interaction laws [14, 95]. This assumes what we would like to discover.

This chapter is a first step toward the automated discovery and validation of macro-scale course grained particle models for polymers. In particular, I seek to match experimentally measured stress-strain curves for three plastics by varying a parametric force law. I also demonstrate hysteresis and material memory based on particle distributions rather than a memory kernel.

6.2 Methodology

6.2.1 Physical Instron

Stress-strain curves for three different plastics were acquired using CBA's Instron.[1] Each test was conducted with a standard ASTM D638 type V coupon [7], 1.6mm in thickness. The three plastics considered were POM (also known as Delrin), HDPE, and PLA.

Their stress-strain curves are depicted in figures 6-1, 6-2, and 6-3, respectively. Some features of these curves are worth mentioning. First, there are a range of magnitudes in both force and extension. PLA and HDPE both exhibit peak forces less than 100 Newtons, while POM exhibits a peak force of nearly 400 Newtons. Meanwhile POM and PLA both fail before 20mm of extension, while HDPE stretches past 100mm before fracturing. Second, the shapes of the curves are qualitatively very different. POM's elastic region smoothly levels out (it's difficult to pinpoint an exact yield strength), and in the plastic region it produces a remarkably constant force of just under 400N from 4mm all the way until it fractures at 12mm. (This indicates that its strain hardening almost perfectly counterbalances its necking.) PLA also demonstrates a long plateau — in this case at about 60N. However it has a distinct peak near its yield strength, reaching a force of roughly 90N. HDPE also has a peak just past its yield strength (almost 50N), but it exhibits more pronounced strain hardening, leading to an upward sloping stress-strain curve for most of its plastic region.

6.2.2 Virtual Instron

To extract comparable data from the simulation, I constructed a virtual Instron. The test coupon is described via an FRep, using the DAGCAD FRep model described in section 3.3. The bottommost and topmost regions of particles are added to driven linear constraints (see section 2.3.1). The bottom constraint simply holds its particles

[1] Thank you Jake Read for acquiring this data.

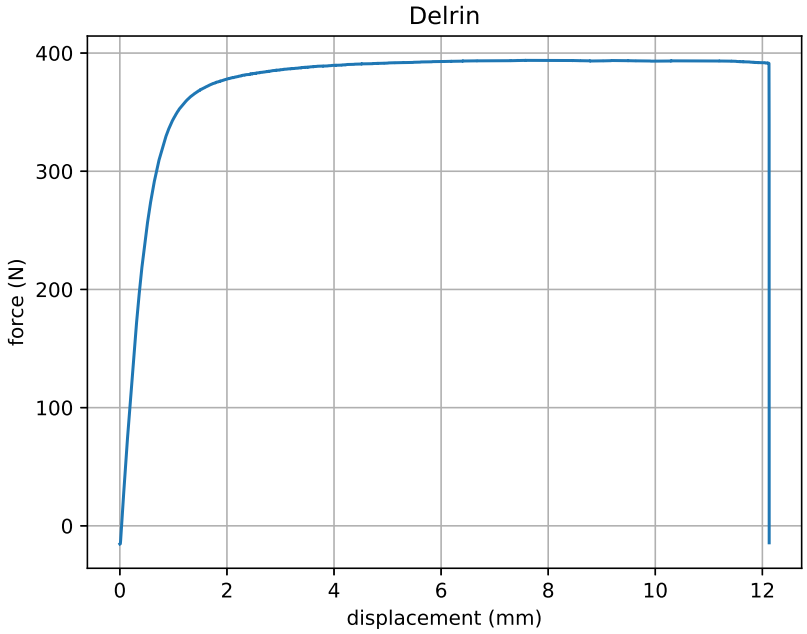

Figure 6-1: Experimentally measured stress-strain curve for POM.

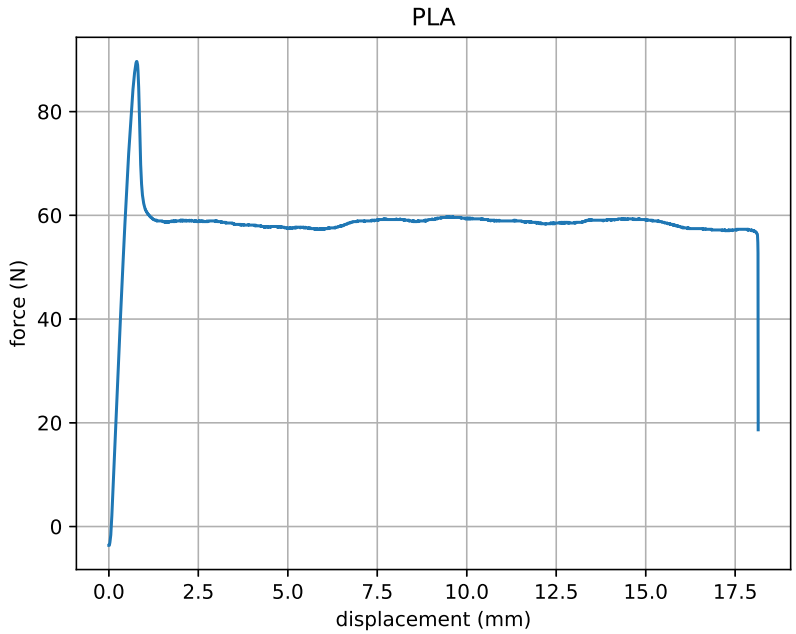

Figure 6-2: Experimentally measured stress-strain curve for PLA.

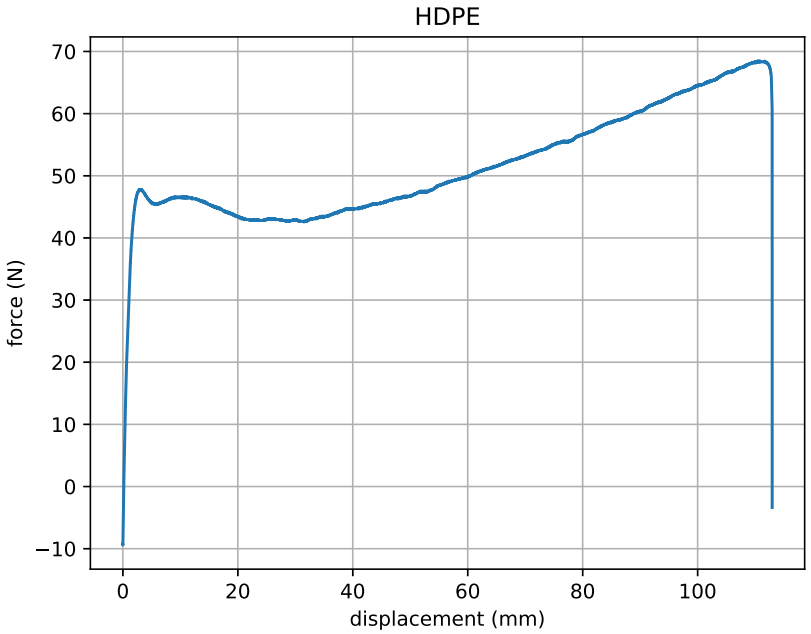

Figure 6-3: Experimentally measured stress-strain curve for HDPE.

in place. The top constraint pulls its particles upward at a constant rate. The bottom constraint measures the net exerted force, while the top constraint measures the average vertical displacement.

At first glance, it can appear that the measurement of force requires some subtlety. The standard definition, if interpreted literally, would suggest that we should intersect the coupon with a horizontal line (or a plane, if the simulation were in 3D), and sum all the forces between interacting pairs of particles that lie on opposite sides of the line. While this is completely feasible in the virtual Instron (if only such instrumentation were as simple in the physical world), it turns out that it's not necessary since simply summing the forces exerted on all the particles in the constraint produces the same result. The area under the stress strain curve up to a certain strain gives the total mechanical energy absorbed per unit volume (or area, in 2D) in reaching that strain. This energy is provided in the form of work done on the coupon by the top constraint, and to capture all the work done we must measure the total force. More intuitively, consider that particles in the interior of the constraint (i.e. completely surrounded by other constrained particles) will have to be pushed much less by the constraint than

those near the boundary, since their neighbors will already be directing them in the right direction.

To aid development and promote exploration, I developed a graphical user interface for the virtual Instron, pictured in figure 6-4. It consists of a live visualization of the coupon, colored to show internal stress, and a floating window with a variety of readouts and controls. The visualization colors each particle according to the sum of the magnitudes of the forces it experiences. Future versions could depict von Mises stress or other scalar metrics.

The floating window shows readouts of various performance metrics and error conditions. It also enables realtime editing of the parameters of the simulation. This includes parameters such as the time step and the amount of dissipation, as well as the coefficients used for the control law of the constraints (see section 2.3.1). More interestingly, it displays a graph the force law, and enables the user to change it. The form of the force will be discussed in section 6.2.3.

Finally, the user can load reference datasets and run extension experiments. The resulting stress-strain curves are plotted in realtime. Controls allow the experiment to be paused or restarted. The score used for optimization of forces laws is also depicted. The exact form of this score is discussed in section 6.2.5. In addition to extension experiments, there is also a stability experiment, which tests the basic viability of the force law. This experiment is also discussed in section 6.2.5.

6.2.3 Representation

The force law specifies a force as a function of distance. All forces between particle pairs are symmetric and act radially. There is no memory in the particles (just their positions and velocities) or the force interactions. These are very limiting assumptions that will be removed in future iterations, but even this basic case proved very worthy of exploration.

The force law is represented as a piecewise linear function. All the studies described in later sections used 5 segments, but this is a compile time parameter that can be changed at will. Conceptually, a force law with n segments is determined by

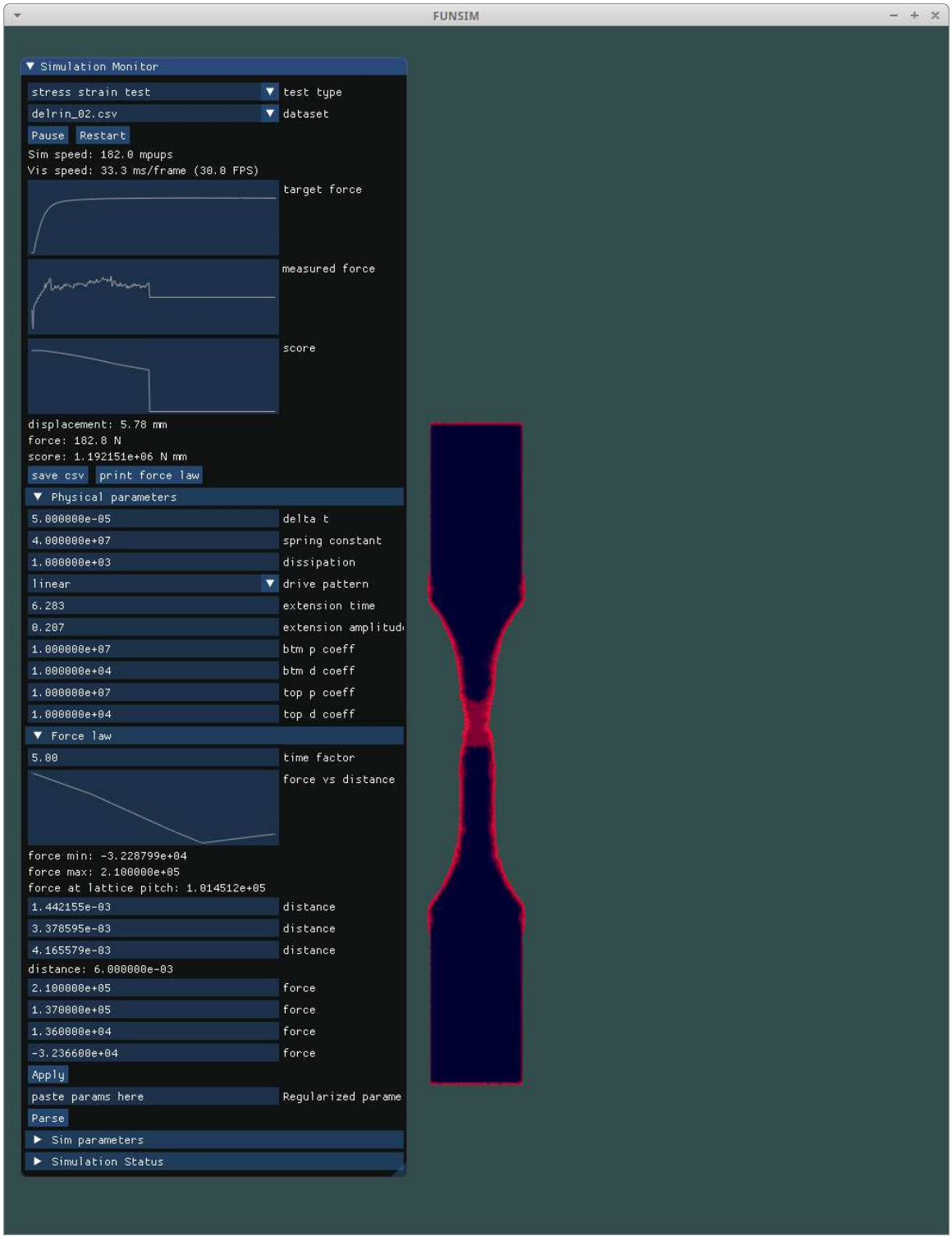

Figure 6-4: A screenshot from the virtual Instron GUI.

$n + 1$ points. The first point is constrained to have an x value of 0, while the last point is constrained to have an x value of the force cutoff distance and a y value of 0 (for more on constraints, see section 2.3.1). The latter constraint ensures that there isn't an abrupt change in force (i.e. jerk) as particles transition from interacting to non-interacting. All together then, for a force law with n linear segments, there are $2n - 3$ free parameters ($n - 2$ positions and $n - 1$ forces).

To be well defined, the other points must have x values between 0 and the cutoff distance. This constraint is easy to satisfy when manually entering values. But enforcing it during optimization would require a constrained search. To avoid this, I use a transformed representation of the position coordinates for the optimization representation. Rather than specify literal x coordinates, this representation specifies the relative distances between the points. This ensures that all tested force laws are at least meaningfully interpretable.

6.2.4 Units

So far we have not specifically stated any system of units to be used. All values of physical parameters have been chosen more for numerical convenience than alignment with real-world systems of measurement. To match a real-world stress strain curve, we will have to remedy this.

This can be done one dimension at a time. The unit of distance is the easiest to nail down. The coupons we used in the real world are 63.5 millimeters long, so this defines our length scale. Rather than mandate that internal simulation numbers must be expressed in mm, or in relative coupon lengths, I simply record the scaling factor and use it to transform distances before they are reported to the user. This conversion factor is approximately 1.575×10^{-2} simulation distance units per millimeter. Mass is also unambiguously defined: the total mass of the particles in the simulation must equal the mass of the real coupon. As will be discussed shortly, this gives us a conversion factor of 3.39×10^{6} simulation mass units to one kilogram. To simplify computations, within the simulation each particle is taken to have a mass of one.

Time is trickier to pin down. From a physical standpoint, it may make the most

sense to define the units of time based on the speed of sound in a material. But other physical processes also have characteristic time scales, which could be used instead. The rate of plastic flow, for instance, is a crucial process for stress-strain experiments. Finally, the rate at which the coupon is extended also defines a time scale. Unfortunately, the dynamic range of time scales in real materials is much higher than we can accurately simulate and still produce results in a timely manner. So choosing the fastest processes (such as speed of sound) is not practical. As a result, I selected an intermediate time scale factor of 0.1 simulation time units to one second. The precise value of this parameter is not ultimately consequential, as the optimization procedure will attempt to find a force law that makes this choice reasonable.

With scales for distance, mass, and time defined, force follows naturally. In particular, our unit of force is simply our unit of mass multiplied by our unit of distance, divided by the square of our unit of time. This yields a conversion of 5,342 simulation force units to one Newton.

Now that our base units are defined we can discuss the other important parameters of the simulation in terms of them. In the experiments presented here I place particles on a hexagonal lattice with a spacing of 0.002 simulation distance units. We now know that this corresponds to 127 micrometers. There are 33,925 particles within each test coupon; each one has a mass of 295 micrograms. (The mass conversion factor was set by assuming a mass of 10 grams for each coupon. In the real world each coupon has a slightly different mass.) I use a time step of 5×10^{-5} simulation time units, or 5×10^{-4} seconds. Thus there will be 2,000 time steps per second. The extension rate is 1.36×10^{-2} simulation distance units per simulation time unit, or about 87 micrometers per second. This is intentionally somewhat faster than the real life experiments, which used a rate of 25 micrometers per second.

6.2.5 Evaluation

The primary objective function is the integrated squared error between the reference stress-strain curve and the simulated stress-strain curve. To facilitate the calculation

of this metric, I load the reference stress-strain curve up front and collect force samples at the same extensions recorded in the physical Instron data.

Data in the virtual Instron is taken up to the extension at which the real plastic coupon failed. Thus a force law will be penalized if a virtual coupon fails too early (since it will measure zero force when real experiment did not), but not necessarily if it fails too late (since we will stop collecting data before it happens). I deem the latter type of inaccuracy acceptable since my primary aim is to reproduce the shape of the curve, rather than precisely model the failure point. After all, the real failure point can vary from sample to sample, and the integrated squared error in the region of disagreement (i.e. where the virtual coupon has failed, but the physical one has not, or vice versa) could easily dominate the error from the region of agreement (in which both coupons have not failed).

But this metric alone isn't sufficient. Most force laws (when sampled uniformly from the representation space described in section 6.2.3) do not produce reasonable physical behavior. In fact, most are wildly unstable, and lead to spontaneous explosions or implosions within the first few steps of the simulation. Needless to say, this is not typical behavior for POM, PLA, or HDPE. Luckily it is easy to detect when this happens, because it causes many particles to leave the bounds of the simulation. So in this case the objective function is computed as a sum of three terms. First, there is the integrated stress-strain curve, as before, assuming a force of zero for all time after the explosion. This ensures that unstable matter can never be given a better score than no matter at all. Second, there are two explosion penalty terms. The first is linearly decreasing in the amount of time the simulation runs without exploding, and the second is larger but exponentially decreasing. The first penalty term ensures that even late stage explosions are penalized, while the second term provides a strong signal to avoid instantaneous explosions.

These explosion penalties help steer the optimization algorithm toward stable matter, but it turns out they still allow plenty of non-physical behavior. In particular, the equilibrium density of the material may not be the initial density (i.e. of the hexagonal lattice). This leads to artificial expansion or contraction of the material

as soon as the simulation begins. To address this I added a second independent test, in addition to the stress-strain measurement. The initial conditions for this test are identical, but rather than drive the constraints, I only use them for measuring the displacement of the ends of the coupon. The simulation is run for a short amount of time: 1800 steps, or 0.9 seconds. The score is computed based on the maximum absolute average displacements of the particles in the top and bottom constraints. A score of zero is awarded if the coupon is completely stationary, while a very large score is assigned if the coupon expands or contracts significantly. The same explosion penalty is also applied, to ensure that unstable matter is still scored poorly even if it explodes before any displacement measurements are taken.

6.3 Results

6.3.1 Force Laws

Figures 6-5 and 6-6 compare simulated and real stress-strain measurements for the best found force laws. Results for HDPE are not reported, as this model has not yet successfully converged. The simulated stress-strain signals are low pass filtered to remove artifacts of the material's resonant vibrational modes. Although the time scales are different, the mass of the physical Instron itself performs a similar function.

Figure 6-7 depicts the convergence of the optimization procedure for both materials. Finally, figure 6-8 shows the optimized force laws.

The virtual stress-strain curves display many of the important features of their physical counterparts. In particular, POM rises at the correct rate, and smoothly transitions to a roughly constant force. PLA also rises at the correct rate, and displays a prominent initial peak.

There is very visible noise in the virtual stress-strain measurements that is not physical. This is an artifact of the discrete nature of the simulation. Specifically, plastic deformation in the simulation occurs via dislocation of particles. These particles tend to quickly slip from one local potential minimum to another, leading to short

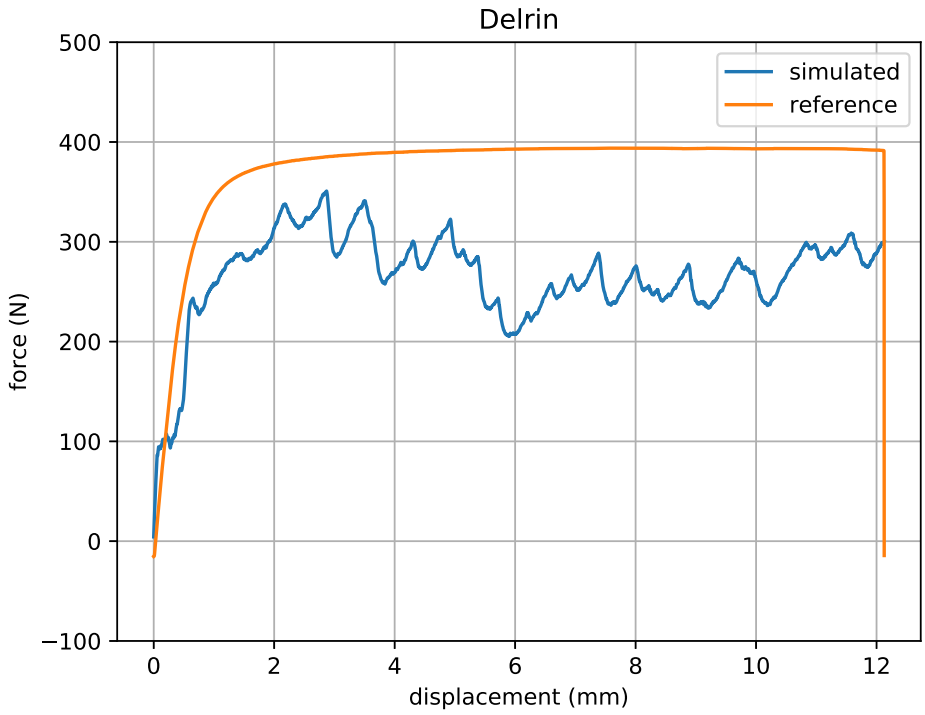

Figure 6-5: Simulated and measured stress-strain curves for POM.

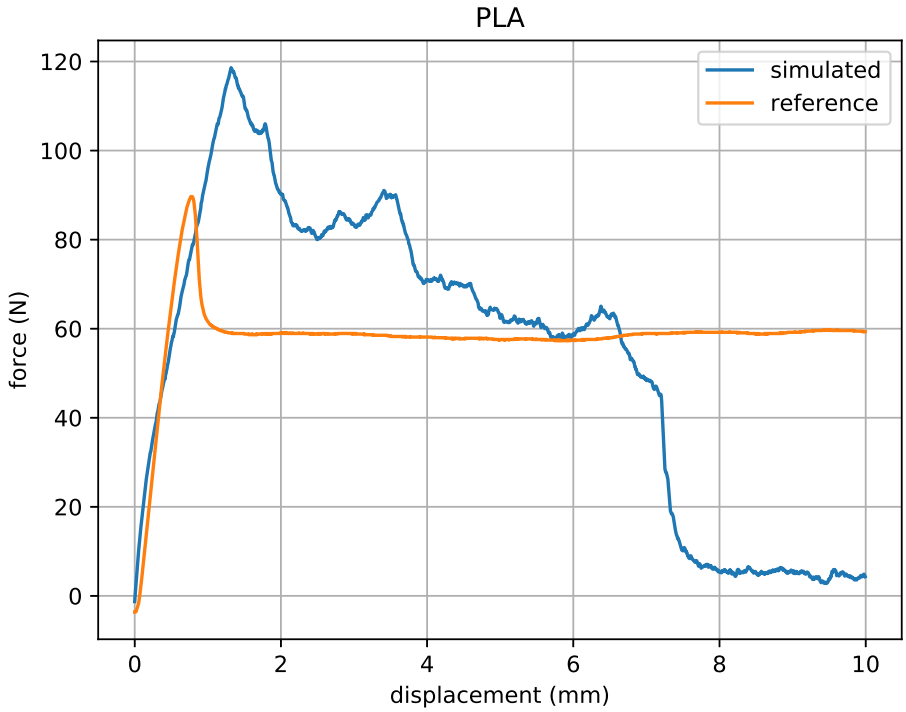

Figure 6-6: Simulated and measured stress-strain curves for PLA.

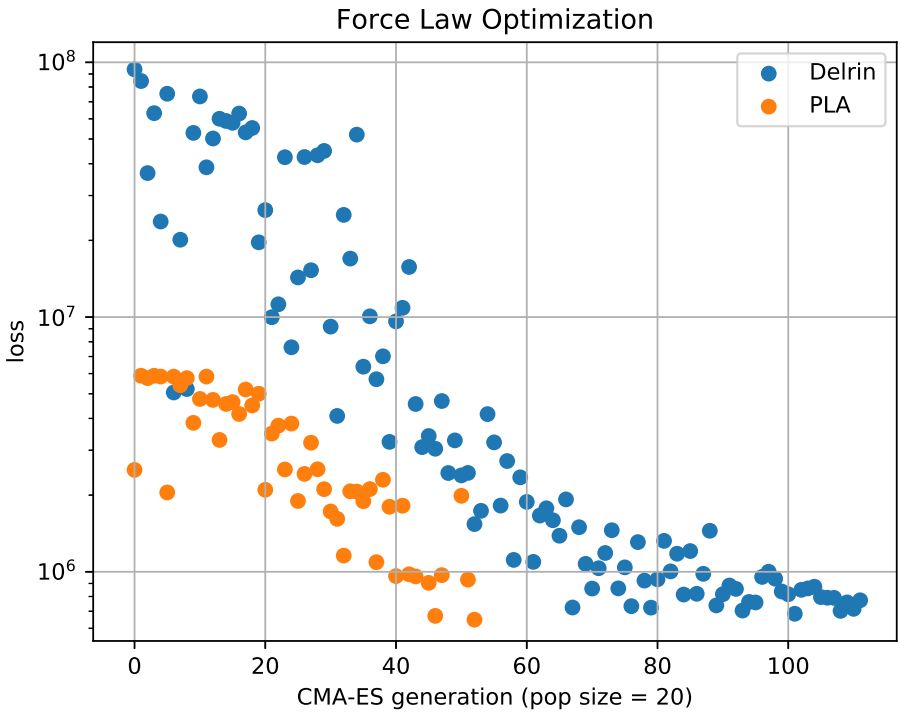

Figure 6-7: Convergence histories for POM and PLA.

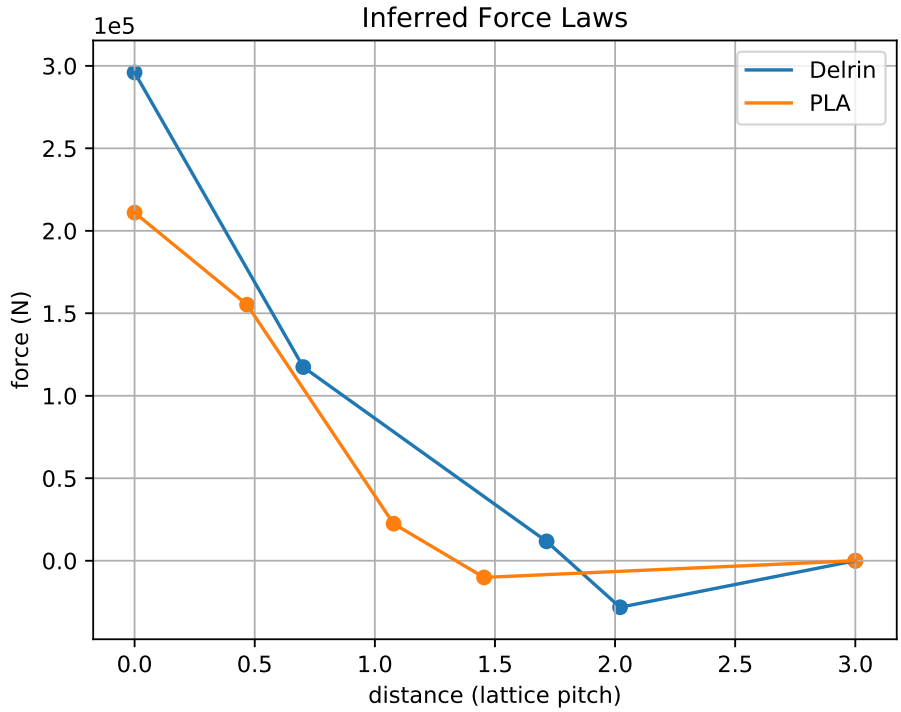

Figure 6-8: Derived for laws for POM and PLA.

bursts of displacement. Cascades of dislocations can occur, amplifying the resulting signal. This process isn't inherently unphysical. Indeed, dislocations in crystal structures are of great theoretical interest, and simulations are developed specifically to study them [52, 68, 51]. However in common materials the dislocations occur at the atomic or molecular level, far smaller than the micrometer scale dislocations that occur in this simulation.

6.3.2 Memory

Interestingly, the resulting materials exhibit memory, despite the fact that neither the particles nor the force laws have any memory themselves. Thus the material memory in these simulations arises purely from the distribution of particles in phase space. This is entirely physical: neither classical physics nor quantum mechanics[2] contains any memory itself. So it is common knowledge that material memory results from particle distributions at the atomic and molecular scale. At larger scales, memory arising from distributions has been studied in granular materials, ranging in size from sand to boulders [31, 28]. But there is a scarcity of literature on non-nanoscale distribution based material memory in non-granular materials. So it is somewhat surprising that material memory in a plastic can be realized utilizing a micrometer particle description. The results of this study are a first step toward characterizing this intriguing possibility.

As a concrete example of how memory can arise from particle distributions, figure 6-9 visualizes the force distribution within two virtual coupons that have been stretched until failure. In particular, the image is colored based on the sum of the magnitudes of the forces acting at each point. It is easy to spot regions that have experienced higher strains.

As a further demonstration of memory, I performed a study on hysteresis with the optimized force laws. Methodologically, it required a simple change to the virtual

[2]Note that quantum entanglement, which is sometimes discussed as if it were a form of memory, simply arises from multi-particle wave functions that cannot be factored into a product of the wave functions of the individual particles.

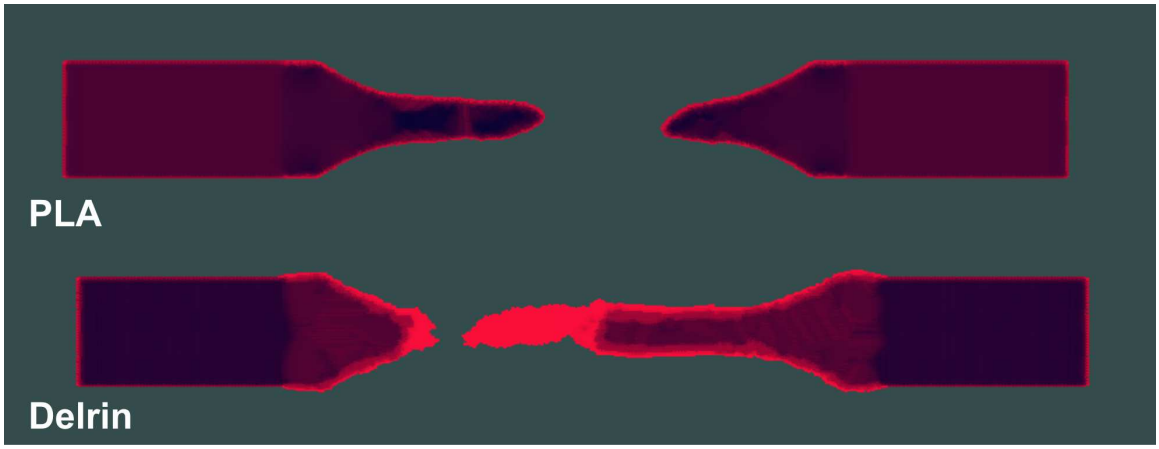

Figure 6-9: Qualitative visualization of internal stresses in two simulated plastics.

Instron GUI. Specifically, I drove the constraint driver for the top of the coupon with a sinusoidal function rather than a linear one. And I modified the data collection routine to collect force measurements at regular time intervals, rather than at the displacements recorded in a reference dataset. Figures 6-10 and 6-11 depict the results.

6.4 Future Work

These initial results suggest many future avenues of research.

To begin, it would helpful to find a way to damp the noise in the virtual stress strain curves. One potential solution is simply to add more particles. This would reduce the Poisson noise that results from random individual displacement events. However it would also increase computational costs. Increasing the damping in the system could also help, although this would require reducing the time step to maintain stability. Finally, it could be beneficial to add non-radial damping. In particular, if there were a strong damping force that decelerated particles as they rotate relative to one another, it could smooth out the contributions of each individual dislocation event.

This last strategy is also interesting apart from considerations of noise. Molecules and even atoms are perfectly capable of exerting torques and shear forces on one

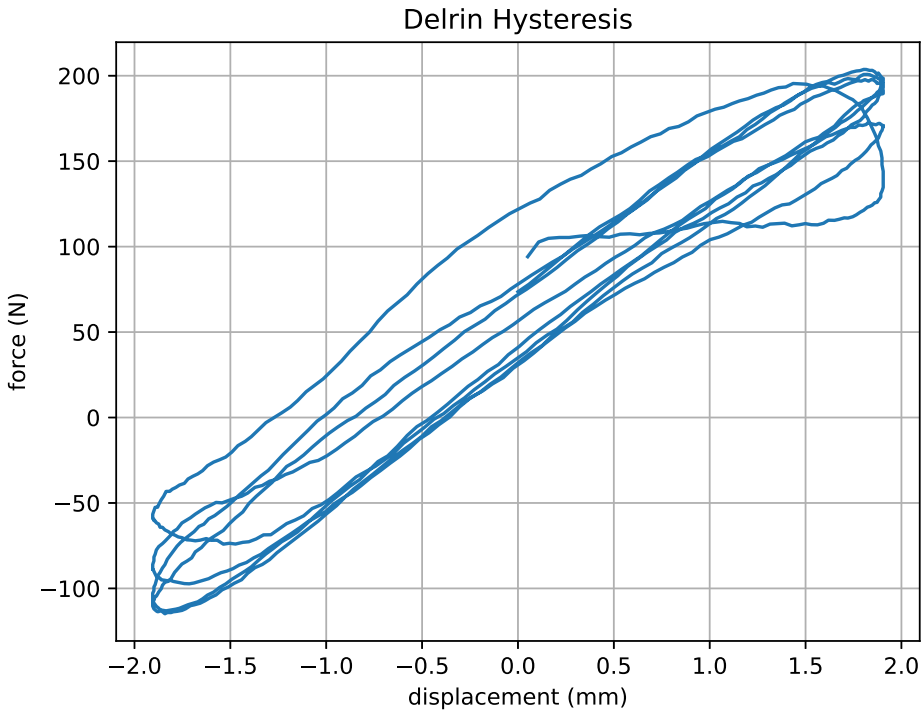

Figure 6-10: Simulated hysteresis curve for POM.

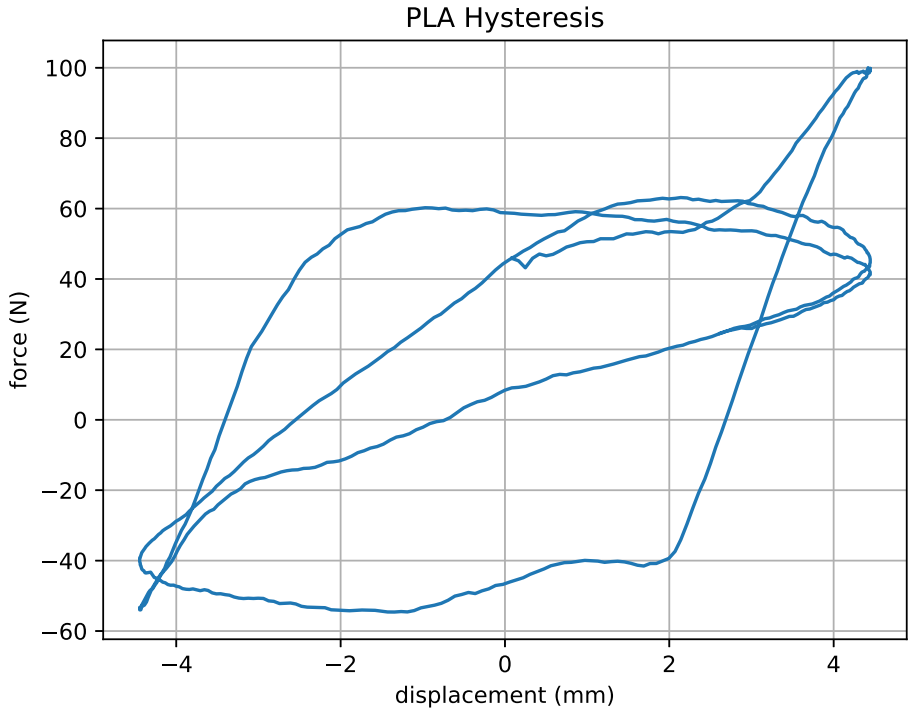

Figure 6-11: Simulated hysteresis curve for PLA.

another. (A simple example of the latter is a dipole-dipole interaction.) Currently this is not possible at the particle level in this simulation. Some course grained models in the literature require non-radial interactions [5], which can be taken as evidence that this would open the door to qualitatively new phenomena.

In the spirit of letting the optimization routine fully determine the force law, I am particularly interested in adding generic state parameters to the particles. This would enable the particles to have a sense of memory, but it would let the best use of that memory be discovered by the optimization algorithm. In particular, in addition to mass, position, and velocity, each particle could posses some small number of additional scalar values. The interaction law would be reframed to accept this additional state as input, and specify how this state should be updated. So the positions and velocities would evolve according to Newton's second law, while the generic particle states would evolve according to a learned update function. Such an interaction rule could be specified as a series of linear transformations and simple nonlinear functions, as in an artificial neural network.

Another exciting possibility for the interaction laws is the introduction of hierarchy. Most of the interesting physics happens near the boundaries of interacting bodies; the bulk material is relatively easy to model. So it seems feasible to use a relatively large number of small particles near boundaries, for both geometric and physical fidelity, and a relatively small number of large particles in the interior regions, to save on compute effort where it is not needed. The critical concern would be developing a force law that interpolates between these different scales while providing consistent material properties. This challenge seems well suited to the automated methods demonstrated in this chapter.

As mentioned in other sections, I am eager to investigate the effectiveness of gradient based optimization techniques.

Finally, I would like to apply these techniques to practical problems in fused filament fabrication (FFF). The prospect of recycling old prints by melting them into new filament has received considerable attention. This is primarily due to the potential resource and energy conservation [96, 50], but certain technical benefits

have been reported as well [91]. However there are significant difficulties. Often the mechanical performance of recycled parts is subpar [20], and the different types of plastic must be kept separate [74, 42]. Using the techniques demonstrated in this thesis, it could be possible to quickly develop models of prints from mixed polymer filament, even if the proportions and ingredients of the filament are not known.

Chapter 7

Evaluation

7.1 Background

In the previous two chapters I presented applications of inverse methods over particle systems: one for design, and one for simulation. They both use the same simulation code and optimization algorithm. Their primary representations encode very different information, but both can be expressed in the same framework so this difference is largely abstracted away. Their evaluation layers, however, are more distinct. This isn't coincidental. At the end of the day, the evaluation metric is what defines the problem.

But despite the lack of theoretical or implementation similarities, there are common themes in the practice of designing tractable objective functions. This is often more an art than a science. Specifically, optimization algorithms are good at coming up with surprising solutions. Sometimes these are welcome, but often they satisfy the letter of the objective but completely miss the spirit. In these instances it is necessary to revise the objective function, to try to steer the optimization algorithm toward more fruitful (or simply practical) regions of the design space.

In the following sections I discuss results of this nature from the applications presented in chapter 5 and chapter 6. Please refer to these chapters for context.

7.2 Gear Design

Depending on the parameters of the simulation, gear teeth may not be necessary at all. Figure 7-1 depicts a set of friction wheels that resulted from an early iteration of the objective function. In this case, the fact that the optimization algorithm converged to such a design was an indication that the material compliance was too high, and the applied load (i.e. the damping force on the right gear) too low. Interestingly, there are regions of parameter space in which CMA-ES sometimes converges to gears, and sometimes to friction wheels, indicating that multiple local optima exist.

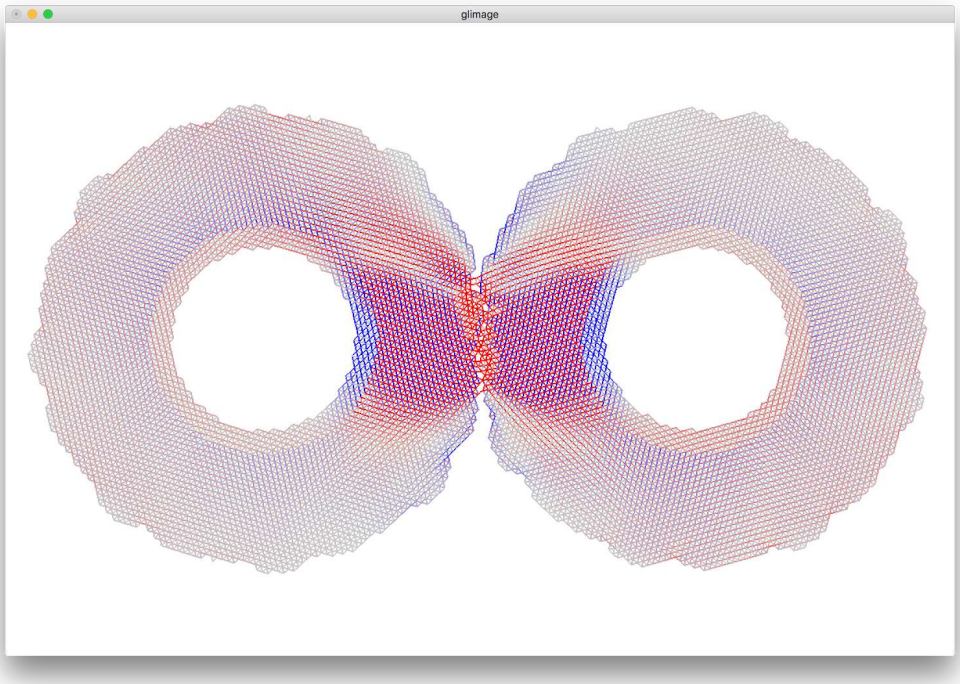

Figure 7-1: A friction wheel is a viable strategy if the material compliance is too high, or the applied load is too low.

Later, I found that I had decreased the compliance of the material too much. Or more specifically, that I was using too large a time step for the newer, stiffer force law. This led to many otherwise reasonable gear designs failing due simulation instability. Some designs rather cleverly avoided this issue by introducing compliance through geometry, as depicted in figure 7-2. In particular, this design effectively introduced flexures on the ends of the gear teeth, so that when they came into contact they

could deflect. This reduced the magnitude of the forces experienced at the gear tooth boundaries, thus reducing the chances of initiating a catastrophic breakdown of stability. This design illustrates the sometimes surprising creativity of the optimization procedure in working around limitations — unfortunately in this case the limitations were accidentally imposed and did not lead to useful innovation.

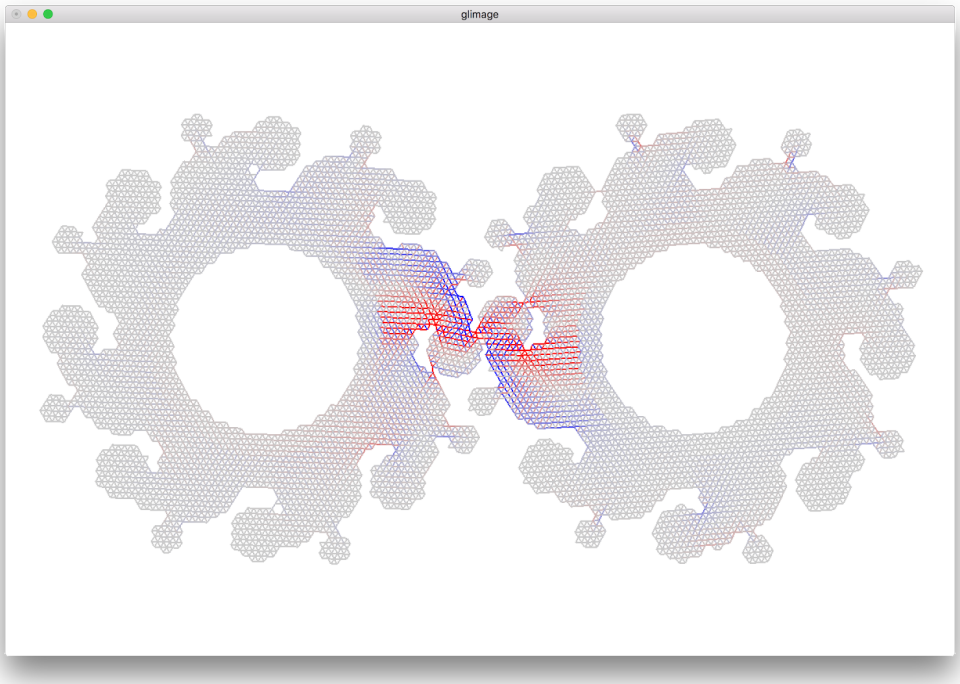

Figure 7-2: An ambitious timestep led to material instabilities — unless compliance was a feature of the design.

With the physics tuned for appropriate rigidity and stability, recognizably gear-like solutions became more common. Still, a number of issues remained. As introduced in section 5.1.2, subtle asymmetries in the objective function can lead to obvious geometric asymmetries. Figure 7-3 depicts one such design, which is not equally effective when driven clockwise versus counterclockwise.

Beyond the symmetry of the objective function, I also varied the primary measurement. Before settling on the mean squared error in angular position, I tried a variety of metrics including mean power, and mean squared power. The latter produced one of the most surprising results. Because of the nonlinearity of the metric,

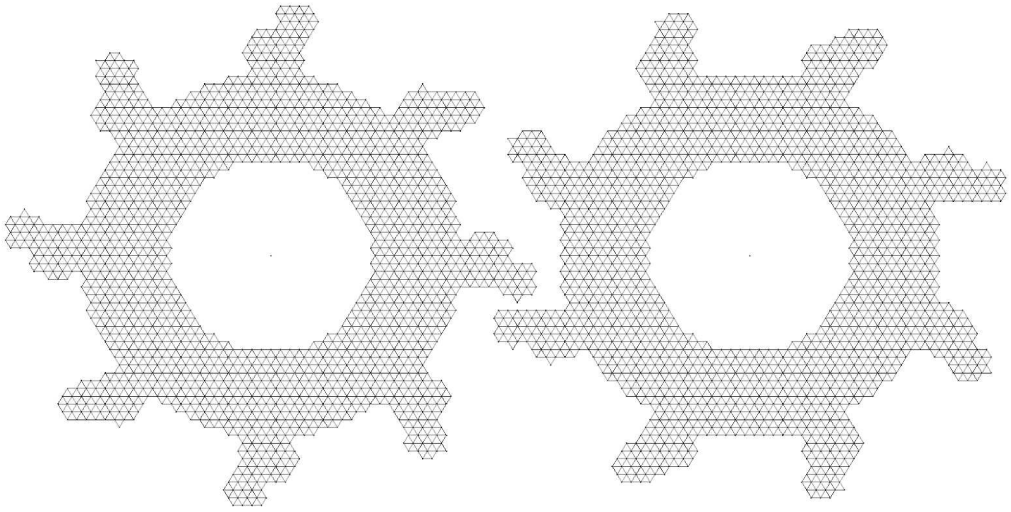

Figure 7-3: Subtle asymmetries in the objective function can become manifested in designs.

gears that transmit power in short bursts are advantaged over those that transmit it consistently, even if the total work performed is equivalent. Clearly this is disadvantageous, which became all too obvious upon seeing the design depicted in figure 7-4. These gear teeth have small ledges, which catch their neighbors as they start to engage. This stores potential energy in the compression of the caught tooth, which is all released at once when the tooth finally slips off the ledge.

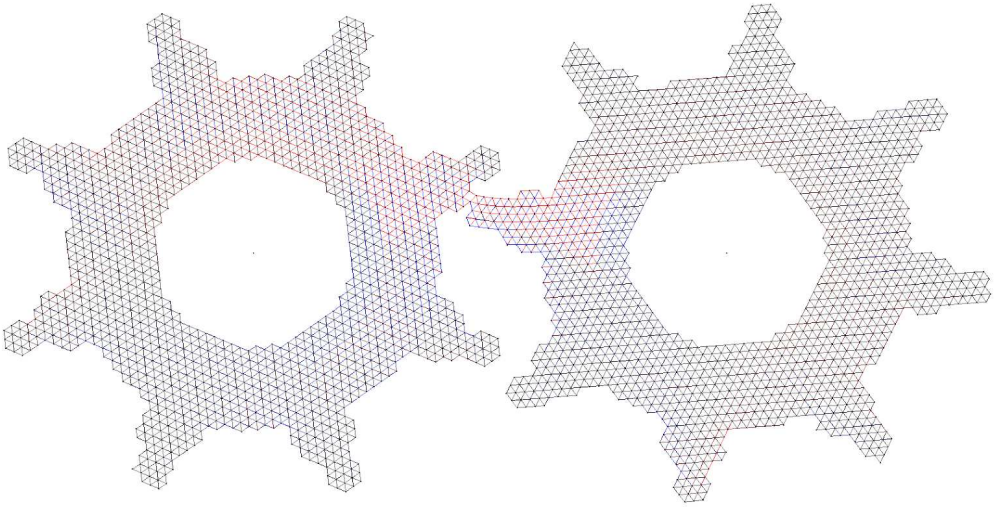

Figure 7-4: Gears designed to catch and release in momentary bursts of power.

7.3 Force Law Search

A similar series of surprises guided the development of the evaluation function for the virtual Instron. As mentioned in section 6.2.5, most force laws do not describe stable physics for solid materials. These laws lead to immediate spontaneous explosion (or implosion), as depicted in figure 7-5. With a uniform penalty for explosions, CMA-ES can in theory find stable regions of force law space through brute force sampling. In practice, it is much better to add a penalty that guides CMA-ES to these useful regions more directly.

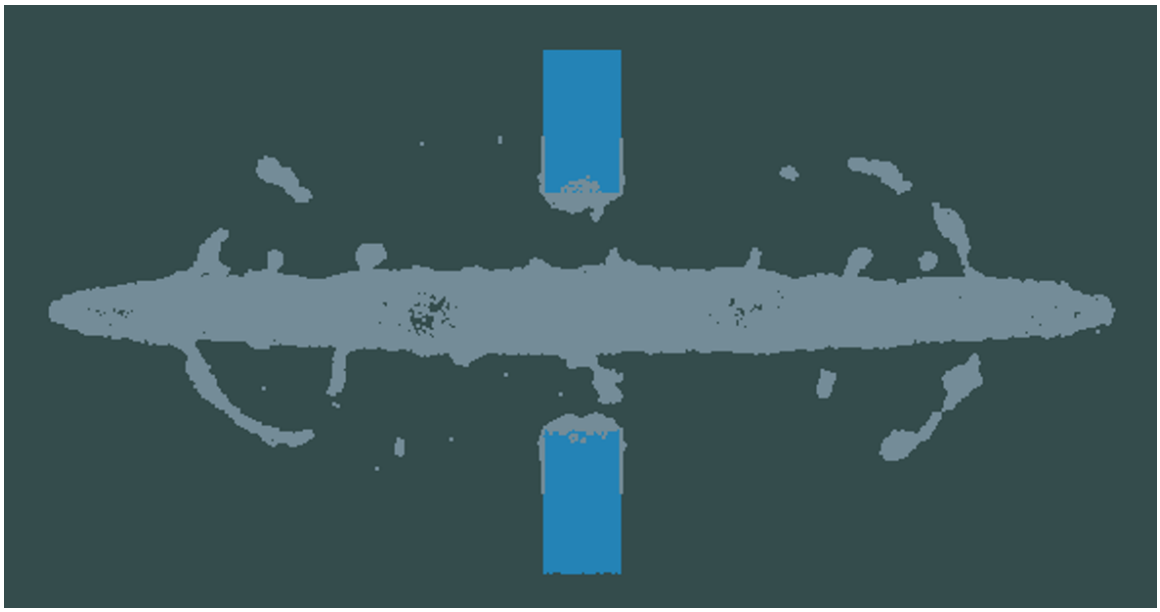

Figure 7-5: The result of an unstable force law.

Even among the stable force laws, the initial arrangement of particles may not be in a local potential energy minimum. This usually leads to rapid expansion or contraction in volume (see figure 7-6), distinguished from the explosions and implosions since the particles do settle into a bound equilibrium state. From the singleminded perspective of matching a stress strain curve, this initial settling can be immaterial. Specifically, though it may produce a large force as soon as the simulation begins (since the clamps must resist the expansion or contraction forces), this force is transient and thus adds only a small amount of error to the stress strain loss. But considering the broader goal of accurately reproducing real material behavior, this

phenomenon is clearly not desired. As a result, the stability test discussed in section 6.2.5 was devised.

Figure 7-6: From left to right: a coupon that has undergone contraction, a nominal coupon, and a coupon that has undergone expansion.

7.4 Conclusions

For both the applications presented in this thesis, developing the final objective function required an iterative process. Along the way, the optimization procedure would produce seemingly counterintuitive results that, ultimately, are guided by what the objective function is, rather than what it is intended to be. Only after several refinements and corrections to the objective are the intent and the reality aligned, allowing the optimization procedure to produce useful results. This finding is consistent with examples from a diverse array of other fields [56].

As a result, the development of a useful objective function still requires manual work on the part of a skilled practitioner. Developing the right objective often requires a solid background in basic physics, and in the relevant specific engineering

discipline. For this reason, it would be inaccurate to claim that inverse methods totally automate design or simulation development. They shift the focus of the skilled engineering labor, and often reduce the amount required, but they do not eliminate it. In summary, knowing the right problem to solve has always been essential. With inverse methods, it is almost everything.

Chapter 8

Conclusion and Future Work

This thesis is a first step toward realizing the underdeveloped potential of inverse methods over physical simulation in engineering disciplines. I presented a particle simulation environment that is free from the limitations of common mesh based alternatives, and can achieve speeds of over one billion particle updates per second. I discussed its implementation on CPU, GPU, and physically reconfigurable networks of microcontrollers, as well as ongoing work toward an FPGA version and a superconducting ASIC. I presented a framework for developing custom design languages that offers seamless integration with optimization frameworks. I discussed the strengths and weaknesses of gradient based and gradient free optimization algorithms in the context of simulation based inverse methods. Collectively, this technology stack is a complete toolkit for defining and solving inverse problems.

Two applications of these tools were presented. The first entailed the design of gear tooth profiles. Within the bounds of this study, existing best practices for gear design were recovered and surpassed. The second entailed the derivation of force laws for course grained simulations of plastic deformation. This application opens the door to much more rapid development of new simulation tools than is currently possible, and demonstrates that material memory can be encoded in particle distributions rather than explicit memory kernels. Finally, I discussed commonalities from the development of these two applications as they relate to the specification of objective functions within inverse methods over particle systems.

All the tools and techniques introduced in this thesis are still being improved in the course of my research at the Center for Bits and Atoms. In the context of this larger arc, I hope that this body of work is only the first step. There are many exciting directions to explore.

The simulation and representation tools have so far been guided by exploration. Now that their utility has been demonstrated on a few different applications, it is time to solidify architectural decisions and expand their scope and generality. For the simulation tools, an obvious first step is expanding to three dimensions. I also plan to add new physics modules. First on the agenda are hydrodynamics, which will draw inspiration from SPH, and thermodynamics. The latter will be an interesting case study in its own right: is it possible to represent temperature in the phase space distributions of meso or macro-scopic particles, as I have shown is possible for material memory? Or will it be necessary to introduce additional particle state? Beyond these forces, there is electrostatics, electrodynamics, and couplings of all of the above. Finally, to facilitate adoption by other researchers, I aim to develop better interfaces. This entails better documentation of internal and external APIs, as well as creating more functional and polished graphical user interfaces.

A related endeavor is the development of custom hardware. I will continue working on conventional DICE: improving the firmware, working toward feature parity with the CPU/GPU simulations, and taking advantage of automatable physical reconfigurability as it is developed. I will also continue working on the Verilog implementation. Comparing CPU/GPU performance to datacenter scale FPGAs will be interesting on its own, but even more exciting long term is the development of ASICs for particle systems. As mentioned in section 2.3.5, CBA is currently working with Lincoln Laboratory to spec out a superconducting chip for this purpose. Based on Josephson junctions, these ASICs could be clocked at 5GHz and burn only 10nW per operation. Even with cooling overhead, projections indicate it could be possible to achieve a 10^5 increase in flops per Watt [27]. Even a small fraction of this theoretically achievable limit would be groundbreaking.

Maintenance of the different simulation implementations is currently too high a

burden, so I am working toward greater code reuse. This is fairly straightforward between the CPU and DICE implementations, since both are written in plain C++. It is harder to share code with the GPU implementation, however, since the necessity of dividing algorithms into separate CUDA kernels imposes an arbitrary and often cumbersome architecture. Verilog is even more foreign. To address these issues, I take inspiration from the Taichi language [40]. I would like to take the abstraction of GPU kernels even further, and introduce a distinction between code that describes an algorithm (expressed in an abstract mathematical form), and code that describes an implementation of an algorithm (expressed in data structures and operations designed for specific hardware).

In the optimization layer, convergence is often still an issue. Encoding more structure in the optimization algorithm itself, such as the cluster based extensions to CMA-ES discussed in section 4.3, is an interesting possibility. Despite the issues discussed in chapter 4, gradient based methods are another. Using these would require making my simulation tool differentiable, a task which I have planned in detail and plan to implement soon. It would also open the door to direct optimization of initial particle positions and velocities — i.e. using an identical representation for representation and simulation.

For applications, to begin there is plenty of work left to do on the two presented in this thesis. Briefly restating the steps described in sections 5.3, I would like to be able to derive different real world gear tooth profiles based on adjustments to the objective function. For instance, if sliding contact is penalized, one might expect to end up closer to an involute gear (which has only rolling contact). As discussed in section 6.4, for the force law studies I want to mitigate the noise in the current systems, and expand to search over more expressive families of interaction laws.

Beyond these applications, my goal is to apply inverse methods over particle systems to increasingly practical and realistic problems. In ongoing work funded by DARPA's Computable Models project [24], we will soon be examining plastic deformation in titanium alloys. This phenomenon can exhibit strong memory effects, and is of great practical importance to NASA. In a different project with NIST, CBA is in-

vestigating low-cost, distributed means of material characterization, as well as online measurement of rheological properties during 3D print jobs in order to optimize material characteristics. Both these tasks involve simulation of some wonderfully messy physics. Finally, CBA is also collaborating with Oldendorff, one of the world's largest operators of dry bulk carriers, to help them meet upcoming emissions guidelines set by the International Maritime Organization. In this project we are fabricating and testing many alternative propulsion systems. All involve fluid structure interaction to some degree, and many require modeling turbulence and interactions with free surface flows. Ultimately the approach described in this thesis must be evaluated on its ability to drive meaningful progress on currently intractable applications.

Bibliography

[1] Niels Aage, Erik Andreassen, Boyan S. Lazarov, and Ole Sigmund. Giga-voxel computational morphogenesis for structural design. *Nature*, 550:84, Oct 2017.

[2] Anurag Ajay, Maria Bauza, Jiajun Wu, Nima Fazeli, Joshua B Tenenbaum, Alberto Rodriguez, and Leslie P Kaelbling. Combining Physical Simulators and Object-Based Networks for Control. In *IEEE International Conference on Robotics and Automation (ICRA)*, 2019.

[3] Anurag Ajay, Jiajun Wu, Nima Fazeli, Maria Bauza, Leslie P Kaelbling, Joshua B Tenenbaum, and Alberto Rodriguez. Augmenting Physical Simulators with Stochastic Neural Networks: Case Study of Planar Pushing and Bouncing. In *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018.

[4] B. J. Alder and T. E. Wainwright. Studies in molecular dynamics. i. general method. *The Journal of Chemical Physics*, 31(2):459–466, 1959.

[5] Davide Alemani, Francesca Collu, Michele Cascella, and Matteo Dal Peraro. A nonradial coarse-grained potential for proteins produces naturally stable secondary structure elements. *Journal of Chemical Theory and Computation*, 6(1):315–324, 2010. PMID: 26614340.

[6] James T. Allison and Daniel R. Herber. Special section on multidisciplinary design optimization: Multidisciplinary design optimization of dynamic engineering systems. *AIAA Journal*, 52(4):691–710, 2014.

[7] Standard test method for tensile properties of plastics. Standard ASTM D638-14, ASTM International, West Conshohocken, PA, 2014.

[8] Josh Barnes and Piet Hut. A hierarchical o (n log n) force-calculation algorithm. *Nature*, 324(6096):446–449, 1986.

[9] Peter Battaglia, Razvan Pascanu, Matthew Lai, Danilo Jimenez Rezende, et al. Interaction networks for learning about objects, relations and physics. In *Advances in neural information processing systems*, pages 4502–4510, 2016.

[10] Martin Philip Bendsøe and Noboru Kikuchi. Generating optimal topologies in structural design using a homogenization method. 1988.

[11] TCN Boekholt, SF Portegies Zwart, and Mauri Valtonen. Gargantuan chaotic gravitational three-body systems and their irreversibility to the planck length. *Monthly Notices of the Royal Astronomical Society*, 493(3):3932–3937, 2020.

[12] Luca Cardelli. Type systems. *ACM Computing Surveys (CSUR)*, 28(1):263–264, 1996.

[13] Michael B Chang, Tomer Ullman, Antonio Torralba, and Joshua B Tenenbaum. A compositional object-based approach to learning physical dynamics. *arXiv preprint arXiv:1612.00341*, 2016.

[14] BM Chaparro, Sandrine Thuillier, LF Menezes, Pierre-Yves Manach, and JV Fernandes. Material parameters identification: Gradient-based, genetic and hybrid optimization algorithms. *Computational Materials Science*, 44(2):339–346, 2008.

[15] Shiyi Chen and Gary D Doolen. Lattice boltzmann method for fluid flows. *Annual review of fluid mechanics*, 30(1):329–364, 1998.

[16] Noam Chomsky. On certain formal properties of grammars. *Information and control*, 2(2):137–167, 1959.

[17] Navier-stokes equation. https://www.claymath.org/millennium-problems/navier Accessed: August 2020.

[18] Georges-Henri Cottet, Petros D Koumoutsakos, et al. *Vortex methods: theory and practice*, volume 8. Cambridge university press Cambridge, 2000.

[19] Charles Augustin Coulomb. *Premier mémoire sur l'electricité et le magnétisme*. Académie Royale des sciences, 1785.

[20] Fabio A. Cruz Sanchez, Hakim Boudaoud, Sandrine Hoppe, and Mauricio Camargo. Polymer recycling in an open-source additive manufacturing context: Mechanical issues. *Additive Manufacturing*, 17:87 – 105, 2017.

[21] CUDA benchmarks. https://browser.geekbench.com/cuda-benchmarks. Accessed: August 2020.

[22] P. A. Cundall and O. D. L. Strack. A discrete numerical model for granular assemblies. *Géotechnique*, 29(1):47–65, 1979.

[23] Ron Cytron, Jeanne Ferrante, Barry K Rosen, Mark N Wegman, and F Kenneth Zadeck. Efficiently computing static single assignment form and the control dependence graph. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 13(4):451–490, 1991.

[24] Computable models (COMP mods). https://www.darpa.mil/program/computable-models. Accessed: August 2020.

[25] Fundamental design (FUN design). https://www.darpa.mil/program/fundamental-design. Accessed: August 2020.

[26] John D Day and Hubert Zimmermann. The osi reference model. *Proceedings of the IEEE*, 71(12):1334–1340, 1983.

[27] Zach Fredin, Jiri Zemanek, Camron Blackburn, Erik Strand, Amira Abdel-Rahman, Premila Rowles, and Neil Gershenfeld. Discrete integrated circuit electronics (dice). Manuscript accepted, 2020.

[28] SJ Friedmann, G Kwon, and W Losert. Granular memory and its effect on the triggering and distribution of rock avalanche events. *Journal of Geophysical Research: Solid Earth*, 108(B8), 2003.

[29] Uriel Frisch, Brosl Hasslacher, and Yves Pomeau. Lattice-gas automata for the navier-stokes equation. *Physical review letters*, 56(14):1505, 1986.

[30] Stephen Gaunt. Hox cluster genes and collinearities throughout the tree of animal life. *The International Journal of Developmental Biology*, 62:673–683, Jan 2018.

[31] Junfei Geng, Emily Longhi, RP Behringer, and DW Howell. Memory in two-dimensional heap experiments. *Physical Review E*, 64(6):060301, 2001.

[32] N. Gershenfeld, B. Schoner, and E. Metois. Cluster-weighted modelling for time-series analysis. *Nature*, 397(6717):329–332, 1999.

[33] J. B. Gibson, A. N. Goland, M. Milgram, and G. H. Vineyard. Dynamics of radiation damage. *Phys. Rev.*, 120:1229–1253, Nov 1960.

[34] Robert Groot and Patrick Warren. Dissipative particle dynamics: Bridging the gap between atomistic and mesoscopic simulation. *The Journal of Chemical Physics*, 107(11):4423–4435, 1997.

[35] Radek Grzeszczuk, Demetri Terzopoulos, and Geoffrey Hinton. Neuroanimator: Fast neural network emulation and control of physics-based models. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, page 9–20, New York, NY, USA, 1998. Association for Computing Machinery.

[36] Ernst Hairer, Ch Lubich, and Gerhard Wanner. Geometric numerical integration illustrated by the störmer-verlet method. *Acta Numerica*, pages 399–450, May 2003.

[37] Nikolaus Hansen, Andreas Ostermeier, and Andreas Gawelczyk. On the adaptation of arbitrary normal mutation distributions in evolution strategies: The generating set adaptation. In *Proceedings of the 6th International Conference on Genetic Algorithms*, pages 57–64, San Francisco, CA, USA, 1995. Morgan Kaufmann Publishers Inc.

[38] Nikolaus Hansen, Raymond Ros, Nikolas Mauny, Marc Schoenauer, and Anne Auger. Impacts of invariance in search: When cma-es and pso face ill-conditioned and non-separable problems. *Applied Soft Computing*, 11(8):5755 – 5769, 2011.

[39] Yuanming Hu, Luke Anderson, Tzu-Mao Li, Qi Sun, Nathan Carr, Jonathan Ragan-Kelley, and Frédo Durand. Difftaichi: Differentiable programming for physical simulation. *arXiv preprint arXiv:1910.00935*, 2019.

[40] Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: a language for high-performance computation on spatially sparse data structures. *ACM Transactions on Graphics (TOG)*, 38(6):201, 2019.

[41] Yuanming Hu, Jiancheng Liu, Andrew Spielberg, Joshua B Tenenbaum, William T Freeman, Jiajun Wu, Daniela Rus, and Wojciech Matusik. ChainQueen: A real-time differentiable physical simulator for soft robotics. In *2019 International Conference on Robotics and Automation (ICRA)*, pages 6265–6271. IEEE, 2019.

[42] Emily J. Hunt, Chenlong Zhang, Nick Anzalone, and Joshua M. Pearce. Polymer recycling codes for distributed manufacturing with 3-d printers. *Resources, Conservation and Recycling*, 97:24 – 30, 2015.

[43] Robert A Jacobs and Michael I Jordan. Learning piecewise control strategies in a modular neural network architecture. *IEEE Transactions on Systems, Man, and Cybernetics*, 23(2):337–345, 1993.

[44] Shinkyu Jeong, Mitsuhiro Murayama, and Kazuomi Yamamoto. Efficient optimization design method using kriging model. *Journal of Aircraft - J AIRCRAFT*, 42:413–420, 09 2005.

[45] Matthew Keeter. libfive. https://libfive.com.

[46] Matthew Keeter. Hierarchical volumetric object representations for digital fabrication workflows. Master's thesis, MIT, 2013.

[47] Matthew J. Keeter. Massively parallel rendering of complex closed-form implicit surfaces. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 39(4), July 2020.

[48] Sebastian Kmiecik, Dominik Gront, Michal Kolinski, Lukasz Wieteska, Aleksandra Elzbieta Dawid, and Andrzej Kolinski. Coarse-grained protein models and their applications. *Chemical reviews*, 116(14):7898–7936, 2016.

[49] Marek Kopicki, Sebastian Zurek, Rustam Stolkin, Thomas Moerwald, and Jeremy L Wyatt. Learning modular and transferable forward models of the motions of push manipulated objects. *Autonomous Robots*, 41(5):1061–1082, 2017.

[50] M.A. Kreiger, M.L. Mulder, A.G. Glover, and J.M. Pearce. Life cycle analysis of distributed recycling of post-consumer high density polyethylene for 3-d printing filament. *Journal of Cleaner Production*, 70:90 – 96, 2014.

[51] Ladislas Kubin, G Canova, M Condat, Benoit Devincre, Vassilis Pontikis, and Yves Bréchet. Dislocation microstructures and plastic flow: A 3d simulation. *Solid State Phenomena*, 23:455–472, 01 1992.

[52] D. Kuhlmann-Wilsdorf. Theory of plastic deformation: - properties of low energy dislocation structures. *Materials Science and Engineering: A*, 113:1 – 41, 1989.

[53] Lammps. https://lammps.sandia.gov/.

[54] Will Langford. *Discrete Robotic Construction*. PhD thesis, MIT, 2019.

[55] Neil Leach. Digital morphogenesis. *Architectural Design*, 79(1):32–37, 2009.

[56] Joel Lehman, Jeff Clune, Dusan Misevic, Christoph Adami, Julie Beaulieu, Peter Bentley, Samuel Bernard, Guillaume Beslon, David Bryson, Nick Cheney, Antoine Cully, Stephane Donciuex, Fred Dyer, Kai Olav Ellefsen, Robert Feldt, Stephan Fischer, Stephanie Forrest, Antoine Frénoy, Christian Gagneé, and Jason Yosinksi. The surprising creativity of digital evolution: A collection of anecdotes from the evolutionary computation and artificial life research communities. Mar 2018.

[57] Wu Li, Luc Huyse, and Sharon Padula. Robust airfoil optimization to achieve drag reduction over a range of mach numbers. *Structural and Multidisciplinary Optimization*, 24(1):38–50, 2002.

[58] Yunzhu Li, Jiajun Wu, Russ Tedrake, Joshua B Tenenbaum, and Antonio Torralba. Learning particle dynamics for manipulating rigid bodies, deformable objects, and fluids. In *ICLR*, 2019.

[59] Zhen Li, Xin Bian, Xiantao Li, and George Karniadakis. Incorporation of memory effects in coarse-grained modeling via the mori-zwanzig formalism. *The Journal of Chemical Physics*, 143(24):243128, 2015.

[60] Zhen Li, Hee Sun Lee, Eric Darve, and George Em Karniadakis. Computing the non-markovian coarse-grained interactions derived from the mori–zwanzig formalism in molecular systems: Application to polymer melts. *The Journal of chemical physics*, 146(1):014104, 2017.

[61] Lennart Ljung. System identification. *Wiley encyclopedia of electrical and electronics engineering*, pages 1–19, 1999.

[62] Miles Macklin, Matthias Müller, Nuttapong Chentanez, and Tae-Yong Kim. Unified particle physics for real-time applications. *ACM Transactions on Graphics (TOG)*, 33(4):104, 2014.

[63] Joaquim Martins and Andrew Lambe. Multidisciplinary design optimization: A survey of architectures. *AIAA Journal*, 51(9):2049–2075, 2013.

[64] Carver Mead and Lynn Conway. *Introduction to VLSI Systems*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1979.

[65] Keith Miller and Robert N. Miller. Moving finite elements. i. *SIAM Journal on Numerical Analysis*, 18(6):1019–1032, 1981.

[66] Joe J Monaghan. Smoothed particle hydrodynamics. *Annual review of astronomy and astrophysics*, 30(1):543–574, 1992.

[67] Cristopher Moore. Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20):2354, 1990.

[68] H. Mughrabi. Dislocation wall and cell structures and long-range internal stresses in deformed metal crystals. *Acta Metallurgica*, 31(9):1367 – 1379, 1983.

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

[70] A. Needleman. Material rate dependence and mesh sensitivity in localization problems. *Computer Methods in Applied Mechanics and Engineering*, 67(1):69 – 85, 1988.

[71] Isaac Newton. *Philosophiae Naturalis Principia Mathematica*. William Dawson & Sons Ltd., London, 1687.

[72] Yew Ong, Prasanth Nair, and Andy Keane. Evolutionary optimization of computationally expensive problems via surrogate modeling. *AIAA Journal*, 04 2003.

[73] Rivka Oxman. Morphogenesis in the theory and methodology of digital tectonics. *Journal of the International Association for Shell and Spatial Structures*, 51(3):195–205, Sep 2010.

[74] Jukka Pakkanen, Diego Manfredi, Paolo Minetola, and Luca Iuliano. About the use of recycled or biodegradable filaments for sustainability of 3d printing. In Giampaolo Campana, Robert J. Howlett, Rossi Setchi, and Barbara Cimatti, editors, *Sustainable Design and Manufacturing 2017*, pages 776–785, Cham, 2017. Springer International Publishing.

[75] A. Rahman. Correlations in the motion of atoms in liquid argon. *Phys. Rev.*, 136:A405–A411, Oct 1964.

[76] David P Rodgers. Improvements in multiprocessor system design. *ACM SIGARCH Computer Architecture News*, 13(3):225–231, 1985.

[77] Eddie Rodríguez-Carballo, Lucille Lopez-Delisle, Ye Zhan, Pierre J. Fabre, Leonardo Beccari, Imane El-Idrissi, Thi Hanh Nguyen Huynh, Hakan Ozadam, Job Dekker, and Denis Duboule. The hoxd cluster is a dynamic and resilient tad boundary controlling the segregation of antagonistic regulatory landscapes. *Genes & Development*, 31(22):2264–2281, 2017.

[78] Vladimir Rokhlin. Rapid solution of integral equations of classical potential theory. *Journal of computational physics*, 60(2):187–207, 1985.

[79] Stanislav Roudavski. Towards morphogenesis in architecture. *International Journal of Architectural Computing*, 7(3):345–374, 2009.

[80] Alvaro Sanchez-Gonzalez, Jonathan Godwin, Tobias Pfaff, Rex Ying, Jure Leskovec, and Peter W Battaglia. Learning to simulate complex physics with graph networks. *arXiv preprint arXiv:2002.09405*, 2020.

[81] Alvaro Sanchez-Gonzalez, Nicolas Heess, Jost Tobias Springenberg, Josh Merel, Martin A. Riedmiller, Raia Hadsell, and Peter W. Battaglia. Graph networks as learnable physics engines for inference and control. *CoRR*, abs/1806.01242, 2018.

[82] Benjamin Sanchez-Lengeling and Alán Aspuru-Guzik. Inverse molecular design using machine learning: Generative models for matter engineering. *Science*, 361(6400):360–365, 2018.

[83] Stephan Schneuwly, Roman Klemenz, and Walter J. Gehring. Redesigning the body plan of drosophila by ectopic expression of the homoeotic gene antennapedia. *Nature*, 325(6107):816–818, 1987.

[84] David E Shaw. A fast, scalable method for the parallel evaluation of distance-limited pairwise particle interactions. *Journal of computational chemistry*, 26(13):1318–1328, 2005.

[85] Ole Sigmund and Kurt Maute. Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6):1031–1055, Dec 2013.

[86] S.A. Silling. Reformulation of elasticity theory for discontinuities and long-range forces. *Journal of the Mechanics and Physics of Solids*, 48(1):175 – 209, 2000.

[87] S.A. Silling and R.B. Lehoucq. Peridynamic theory of solid mechanics. In Hassan Aref and Erik van der Giessen, editors, *Advances in Applied Mechanics*, volume 44 of *Advances in Applied Mechanics*, pages 73 – 168. Elsevier, 2010.

[88] Karl Sims. Evolving virtual creatures. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 15–22, New York, NY, USA, 1994. ACM.

[89] Deborah Sulsky, Zhen Chen, and Howard L Schreyer. A particle method for history-dependent materials. *Computer methods in applied mechanics and engineering*, 118(1-2):179–196, 1994.

[90] Terence Tao. Quantitative bounds for critically bounded solutions to the navier-stokes equations, 2019.

[91] Xiaoyong Tian, Tengfei Liu, Qingrui Wang, Abliz Dilmurat, Dichen Li, and Gerhard Ziegmann. Recycling and remanufacturing of 3d printed continuous carbon fiber reinforced pla composites. *Journal of Cleaner Production*, 142:1609 – 1618, 2017.

[92] Loup Verlet. Computer "experiments" on classical fluids. I. Thermodynamical properties of lennard-jones molecules. *Phys. Rev.*, 159:98–103, Jul 1967.

[93] David W Walker and Jack J Dongarra. Mpi: a standard message passing interface. *Supercomputer*, 12:56–68, 1996.

[94] Kesheng Wu and Horst Simon. Thick-restart Lanczos method for large symmetric eigenvalue problems. *SIAM Journal on Matrix Analysis and Applications*, 22(2):602–616, 2000.

[95] Zhaohui Yang and Ahmed Elgamal. Application of unconstrained optimization and sensitivity analysis to calibration of a soil constitutive model. *International journal for numerical and analytical methods in geomechanics*, 27(15):1277–1297, 2003.

[96] Shan Zhong and Joshua M. Pearce. Tightening the loop on the circular economy: Coupled distributed recycling and manufacturing with recyclebot and reprap 3-d printing. *Resources, Conservation and Recycling*, 128:48 – 58, 2018.