Design for Customized Manufacturing

by

Francisca T. Gil Ureta

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Mathematics

New York University

September 2019

Denis Zorin

Abstract

Over the past few years, 3D printing technology has captivated business and consumers alike with its promise of affordable *custom manufacturing*. The expectation is, in the future, people will be able to easily customize and manufacture objects to fit individual needs. To make this a reality, we need new methods that support the creative process of makers, from conception to fabrication.

In this thesis, I present three projects where we reexamine the tools and workflows used for customized design. The core idea behind these projects is that, compared with traditional methods, we design for an unknown or changeable manufacturing process, which affects the life-cycles of design. Our goal is to create tools that simplify the modification, optimization, and evaluation of designs such that they can be easily altered to fit manufacturing and personal constraints.

Although fabrication constraints are unlimited, we can study specific domains to learn the most common ones. In the first project, we present an interactive modeling tool for designing mechanical objects, which are determined mostly by kinematic constraints. In the second project, we study the structural efficiency of shells and introduce an efficient method for designing shell reinforcements of minimal weight. Finally, in the third project, we develop a robust collision resolution algorithm, crucial for the design and optimization of models subject to dynamic impulses.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Over the last few years, flexible manufacturing technologies, like 3D printing, have captivated businesses and consumers alike with their promise of *custom manufacturing*. The expectation is that, in the future, people will be able to easily customize, designs, and manufacture them on-demand, from a fitted dress, and a unique bicycle, to custom prosthetics. Using digital manufacturing, people will bring to life unique objects that fit their needs.

To an extent, this is something we can do even today. On websites like Shapeways and Thingiverse, we can access unique ready-to-print designs and 3D print them on-demand. However, customizing these designs is not trivial. Even simple algorithms, like resizing or rotating a model, can affect the object's printability. For example, down-scaling a design can make some areas of the model too thin to print, destroying the original functionality of the object. If we need a smaller object, our best option is to go back online and find one that fits our requirements.

Models can be customized in a variety of other ways, e.g., by changing materials or introducing joints, with unpredictable effects on printability and functionality. Ensuring that necessary properties of objects are preserved by hand is time-consuming and prone to errors.

Effectively customizing a model requires exploring a large but constrained design space

(including geometry, material, and color). Traditional design tools for manufacturing (CAD) are primarily designed for conventional mass manufacturing processes, and are often bound by legacy limitations of software and workflow designs. These are typically unsuitable for exploring the broader space of shapes that can be manufactured using more flexible technologies. On the other hand, systems targeting computer animation and games (e.g. Maya, Blender, ZBruch) provide more flexibility in complex shape design, but lack a connection with optimization and physical modeling of resulting objects. Moreover, most conventional software does not include many essential features of customizable product design [28].

In this thesis, we make several contributions to computational tools needed to support customization. Before summarizing these, we briefly discuss the background in customization and additive manufacturing.

1.1 Background

1.1.1 Custom Manufacturing

The concept of custom manufacturing refers to the ability to offer tailored products and services to customers. In practice, customization is achieved by offering options to the customers, which can range from individually designed products to custom packaging. The different levels of individualization have been studied by several authors ([40, 54, 34, 74, 88, 28]). In this thesis, we develop computational tools that support customized fabrication, Level 7 in Da Silveira et al. [28]:"manufacturing of customer-tailored products following basic, predefined designs."

In addition to determining the level or range in which a product can be customized, we must understand how individual make decisions. Hague et al. [37] suggests three answers to this question: (1) the designer decides on a set of predetermined alternatives and the

customer chooses among them; (2) the customer can modify the design using a high-level user interface that requires little training and no technical knowledge; and (3) designer and customer collaborate in real-time. These three points define the range in which the customer can be involved in the design process. What level works best will depend on the type of product created.

Enabling technologies Until recently, the cost of custom manufacturing has been too high for most common products. However, the development of new rapid manufacturing technologies has changed this, making custom manufacturing an affordable option in many more cases, especially for B2B orders. For example, companies like Nike and Adidas allow clients to customize their shoes, both aesthetically and functionally, to fit their own needs [43].

Although advanced manufacturing technologies are essential to enable custom manufacturing, computer-aided design also plays an important role [28]. The former allows designers to manufacture and test various designs without changing the manufacturing pipeline (i.e. rapid manufacturing); on the other hand, the later must allow rapid redesign and accurate analysis.

1.1.2 Customization and Additive Manufacturing

The last decade has seen many advances in manufacturing technologies. Additive Manufacturing (AM) has been one of the most successful ones and within it 3D printing, which many have titled the "new industrial revolution" [16].

AM has substantially decreased the difficulty of manufacturing highly complex shapes, and often allows one to create parts that cannot be produced by any other means. Thompson et al. [94] presents a comprehensive summary of its history and design possiblities. Here we repeat some aspects relevant to this thesis. Compared to conventional fabrication techniques,

AM has the ability to create unique geometric forms at a macro and micro level. At a macro level, AM can be used to create intriguing aesthetics (e.g. jewelry, home furniture, clothing accessories) and complex internal features to increase performance (e.g., optimized fluid channels, acoustic dampening devices, minimum weight structures). At a micro level, AM can create custom surface textures and volumentric metamaterials. Additionally, AM technologies exist for a wide range of materials including polymers, metals, ceramic, sand, clay, concrete, chocolate, and even edibles (sugar, frosting, pasta, cheese, ground beef, and egg whites). To explore this new large and complex design space, we require new approaches to the design process and practice.

Traditional approaches were tailored to design objects which can easily be created using traditional manufacturing processes. This influenced the tools and algorithms implemented in most commercial CAD programs, as well as the techniques taught to designers. Existing CAD software lacks the ability to generate multi-scale geometry or design organic shapes, and engineering simulation software (e.g ANSYS, COMSOL) fails to analyze these complex geometries. The lack of proper tools places much more responsibility on the designer, who must deliver a finished piece satisfying all requirements.

1.2 Contributions

This thesis presents new techniques supporting the goal of seamless exploration of the design space while ensuring certain constraints are satisfied.

We considered several scenarios:

- In Chapter 2, we present an approach to interactive modeling for designing *mechanisms*, i.e., objects consisting of solid parts connected by joints, which are determined at a high level by kinematic constraints. Our goal is to reduce or eliminate the need to manipulate low-level geometry of mechanical joints. Our system automatically infers a

small number of plausible joint configurations, which are offered to the user through an intuitive interface, supporting rapid exploration of the joint design space.

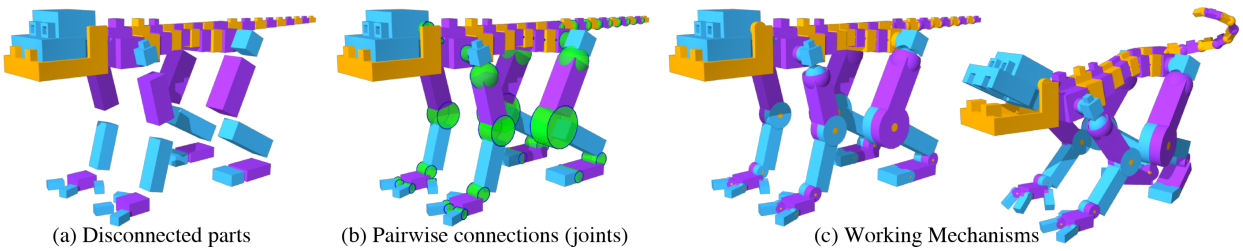
- In Chapter 3, we study the structural efficiency of shells and introduce an efficient method for automatic addition of shell reinforcement structure of minimal weight, that ensure a bound on the maximal stress in the structure. This allows a user to focus on shape design, with structural stability automatically ensured by the added reinforcement structure. Our method allows for a broad range of optimal structures, including previously known ones (Michell trusses, variable thickness shells, and ribs/grillages for plate support). The components of our method include a novel formulation for Mitchell-like continua on surfaces, taking bending into account and a parametrization of the reinforcement structure, supporting both discrete ribs and continuous thickness variation. We also propose an efficient and robust optimization method for the nonlinear and non-convex optimization problem that we need to solve.

- Chapter 4 presents our work on robust collision and contact handling, motivated by the needs of optimal design of shapes for which variable contact is an essential part of their function (e.g., folding shapes, chain mail-like structures and protective packaging made from sheet material). The need to use simulation with contact inside an optimization loop makes robust handling of contact essential. Our approach, which was developed for 2D shapes, is a first demonstration that an essentially unconditional collision resolution is possible for a broad range of time steps. It is controlled by a single parameter ϵ_b which is a trade-off between running time and accuracy; while tuning of other algorithm parameters can improve performance, it is generally not required for it to succeed.

Chapter 2

Interactive Modeling of Mechanical Objects

This chapter is based on the publication Gil-Ureta et al. [33] in collaboration with Chelsea Tymms and Denis Zorin. My contributions include the development of all steps of the algorithm, except the sweept computation. I also designed and executed the user evaluations, and performed the data analysis.



(a) Disconnected parts (b) Pairwise connections (joints) (c) Working Mechanisms

Figure 2.1: Our system enables users to create working, printable mechanisms from a set of disconnected parts by adding pairwise connections (joints). The model shown here has more than 30 hinge and sphere joints added using our system in about 15 minutes. Original model "Blocko" by Mikkel_bf.

2.1 Introduction

Mechanical objects are one of the most ubiquitous types of man-made objects. Many of the objects with which we interact during the day are part of a mechanism, including doors, faucets, coffee machines, cars, desks, computers, lamps, and tools. Unsurprisingly, creating models of mechanical objects is a common task. With standard modeling approaches, this task requires considerable effort and expertise. At the same time, as 3D printing and other technologies for personalized manufacturing become commonplace, the number of novice users creating 3D models, and particularly mechanical models, is rapidly increasing.

Constructing operational mechanical models is often difficult. For example, many of the models offered by Shapeways, one of the largest online services for 3D printing, *look like* mechanical objects but do not include functional mechanical joints. We conjecture that part of the reason for this is that modeling mechanisms, with actual moving parts, is far more difficult than modeling static shapes. The goal of our work is to develop a method to simplify and streamline the creation of basic mechanisms for users with moderate or no modeling experience as well as for more advanced users.

Formally, a mechanism is a structure composed of multiple rigid bodies (*parts*) interconnected at specific areas (*joints*). The geometry of the joint limits the relative motion of the connecting parts, thus defining the behavior of the mechanism [61]. Consider, for example, the object in Figure 2.1: the ball-and-socket joints in the shoulders and the tail allow three-dimensional rotation while the hinges in the knees and fingers rotate about a single axis. As we can see, the kinematic behavior of a mechanism is intimately related to the geometry of the joints. Therefore, to create a good-quality mechanism, an artist must be able to understand how the geometry of a joint affects its motion.

In traditional 3D modeling tools, the geometry and motion of the joints are defined separately. In a typical scenario, the artist first models the geometry of all the mechanism's

parts as independent components. Then, the artist creates a system of virtual joints (*rig*, or skeleton) which defines the kinematic behavior. Finally, the artist bounds each part of the model to a joint, with the part inheriting its relative motion.

We observe several major challenges with applying the traditional modeling workflow to mechanisms. First, the position, orientation, and proportions of the joint need to be computed by the artist, who must ensure the joint implements the desired motion. This is accomplished by a combination of time-consuming experimentation and geometric calculations, which are not always simple [51]. Second, since the behavior of the mechanism can only be tested after the geometry is created and associated with a separately-created skeleton, errors are often found late in the process; parts that look good statically might intersect or become disconnected during motion. And third, finding and fixing these errors is not a trivial task for complex shapes and may require many iterations.

In this chapter, we present an approach to modeling mechanical objects that addresses these challenges. The two main principles that we use are (1) integration of the geometry definition with the rig construction, which defines motion, and (2) inferring the geometry of joints from the geometry of parts semi-automatically, allowing the user to seamlessly explore the space of joint configurations.

The contributions of our work include a semi-automatic technique to find a set of feasible joint positions, orientations and sizes for a pair of parts; a user interface design for exploring these configurations; and a method for adapting part geometry, when necessary, to the desired motion ranges.

We demonstrate our approach by converting a number of example 3D shapes into functional mechanisms with aesthetically satisfactory joints. Our evaluation shows that inexperienced users can efficiently use our interface and that overall performance is significantly better than what can be achieved with traditional modeling tools.

2.2 Related Work

Modern commercial CAD software for designing mechanisms and assemblies is tailored for professional machine designers and mechanical engineers (eg. SolidWorks, SolidEdge, Inventor). To properly use these systems, the user must have a good understanding of mechanics and modeling, which makes them unfit for hobbyists and novice users. In contrast, with our system, a user can create mechanisms, with physically realizable joints, without having to manipulate low-level geometry of joints.

The closest work, in terms of goals, is Koo et al. [51]. This work addresses the problem of creating works-like prototypes of mechanical objects. The goal is the physical realization of a particular functionality defined by high-level functional relations between parts, with focus on converting these high-level relations to low-level kinematics. The work is restricted to parts shaped as cuboids and does not consider user-interaction or aesthetic aspects of joint placement; this makes it possible to use a minimalistic algorithm for joint placement, orientating and sizing.

A similar problem, that of creating articulated characters, is solved by Bächer et al. [9] and Calì et al. [22]. These works describe a pipeline for converting a character shape (typically, a single-component organic shape) into an articulated model by automatic segmentation and joint insertion. Both methods rely, in a fundamental way, on placing the joints in the interior of the object and on the availability of a skeleton or skinning weights to determine joint size and placement (with orientation of hinges in Bächer et al. [9] determined manually). Although our targeted output models share similarities, we target input models consisting of separate parts, as is typical for mechanical objects, and do not assume a skeleton as input. We only require specification of pairs of parts to be connected by joints, with no geometric information as input, and focus on inferring a set of plausible options for joint geometry from part geometry.

Mechanism Design. Several works in graphics are concerned with design methods that facilitate the creation of functional objects. For instance, Hergel and Lefebvre [42] turns a 2D mechanism into a working 3D model, but is restricted to planar mechanisms. Koyama et al. [52] describes a method for automatically creating 3D-printable connectors; however, they use only rigid rod connectors.

Other works propose methods for designing mechanisms from a purely functional point of view: Zhu et al. [107] is concerned with an automatic design and placement of driving mechanisms, attached to an existing articulated character, to match simple motion trajectories created by a user. A related problem is solved in Coros et al. [27], handling a larger space of possible motion trajectories. Thomaszewski et al. [93] and Bächer et al. [10] present a system for the design and editing of linkage-based characters, allowing the creation of characters actuated by a single motor that achieves the desired motion while maintaining aesthetic appeal. All these methods require an existing virtual mechanical character as input, whereas our system creates such a mechanical character as the output.

More recently, Megaro et al. [62] presents a system for the design of mechanical robots. Their approach uses the skeletal structure of the robot as input and adds motors at joint locations. The geometry for the body parts is automatically generated to connect the motors. The user can't easily modify the shape of the parts; augmenting the geometry is allowed, but the user must ensure there is no interference during motion. With our system, the user has greater control of the aesthetics of the model, while never having to manipulate low-level geometry.

Several works, [55, 95, 56], focus on furniture design. In particular, Lau et al. [55] describes a system for furniture assembly based on grammars, including adding parametrized joints from a library. The goal of this work is similar to ours, but it focuses on a narrower domain, which allows the use of grammars and a higher degree of automation. However, this approach restricts the possibilities of joint placement and sizing.

Schulz et al. [81] demonstrates how conceptual sketches can be converted to manufacturable objects by adding the required connectors and joints based on a database of examples. In principle, this type of approach could be viewed as the ultimate goal for the problem we are solving, allowing near-complete automation, but in its current form, it requires a large and precisely-annotated collection of domain-specific examples.

A number of works, e.g. Mitra et al. [64] and Xu et al. [99], consider the complementary problem of estimating joint kinematics from surface geometry. These methods use the geometry of the joint, already present in the model, to determine the nature of the connections using primitive fitting [64] or slippage analysis [99].

Swept Volume. An essential feature of our method is the adaptation of part geometry to allow for desired joint motion, for which sweeps are required. Many general algorithms exist to compute swept volumes, such as Abdel-Malek et al. [2] and more recently von Dziegielewski and Hemmer [97], but these are relatively slow. We use an efficient special-case algorithm described in Section 2.7 to create the sweep envelope, using a method similar to that of Peternell et al. [72].

2.3 Simple Mechanisms and Joints

Mechanical joints between parts are described in terms of kinematic pairs of primitives. In this chapter, we are primarily concerned with *lower pairs*, [29], i.e. joints that constrain a point, line, or plane of one part to a point, line, or plane in another part.

There are six standard kinematic pairs (Figure 2.2):

1. *A revolute pair (hinge)* requires two collinear lines in the connected parts, with no motion along these lines; that is, planes on each part, perpendicular to the aligned lines, maintain contact with each other. It has one rotational degree of freedom.

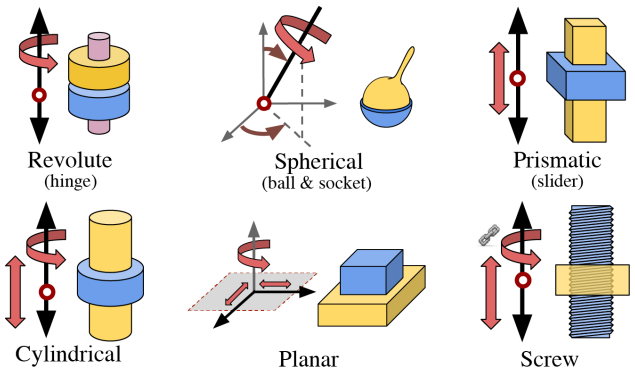


Figure 2.2: Six standard kinematic pairs.

2. *A spherical joint* constrains two points in the connected parts to have the same location. It has three rotational degrees of freedom.

3. *A prismatic joint (slider)*, just as the hinge, requires the alignment of two lines in the connected parts and contact between two planes, but now the planes are *parallel* to the common line. It has one translational degree of freedom.

4. *A cylindrical joint*, like the hinge and slider, requires two collinear lines in the connected parts, but it does not add plane constraints. It has two degrees of freedom, one translational and one rotational, and it can be viewed as a combination of a hinge and a slider.

5. *A planar joint* requires that a plane in one part remain in contact with a plane in the other part. It has three degrees of freedom, two translational and one rotational.

6. *A screw joint* requires that helical lines (threads) on two parts remain aligned; it has one degree of freedom, corresponding to a mutually constrained translation and rotation.

In this chapter, we consider linked assemblies using the first three pairs on the list (hinge, sphere, and slider), but effectively all except the screw joint can be handled: construction of a cylindrical joint is easily reducible to a hinge joint with a free translational axis; and the

plane joint, although rarely used, can be composed by two sliders and a hinge.

While it would be simple to add the screw joint, we have not found many examples of its use; this type of joint is primarily used in linear actuators as a part of more complex mechanisms involving higher pairs (gears or cams), which we do not consider. The principles we developed for interactively placing and determining dimensions of the joints can also be applied to more complex joints, such as gimbal, six-degrees-of-freedom joint, universal joint, etc. All these joints can typically be decomposed into the primitive joints described here.

2.4 Overview

Our approach to modeling mechanical objects is based on several principles:

- The user specifies only the hierarchy of the parts and the type of kinematic pair they form; the kinematic skeleton, which defines part motion, is inferred from the joint type.
- The system identifies part *interfaces* (regions where the parts can be connected) and determines plausible joint *configurations* (consisting of *position*, *orientation*, and *dimensions*). The user interaction is mostly reduced to choosing among a small number of these configuration options and setting ranges of motion.
- The system creates the geometry of the joints and modifies part geometry when needed.

We describe the user perspective of the workflow in greater detail in Section 2.5. A number of algorithms are needed for this workflow; most are either independent of the type of joint or are easily adaptable to different joint types. One key algorithm extracts interfaces from a pair of parts (Section 2.6.1); another determines configurations from the interface geometry (Section 2.6.2). Finally, in Section 2.7, we describe our technique for creating joint geometry and updating part geometry to make the object physically realizable.

2.5 Workflow

The overall workflow is shown in Figure 2.3. To create a joint in our system, the user starts by selecting two parts of the model and specifying the type of mechanical joint that will connect the pair. Based on the selection, our system calculates valid options for interfaces and joint configurations (position, orientation, and dimensions). The user then selects one of the suggested configurations. Using this configuration, our system creates a fabricable version of the design.

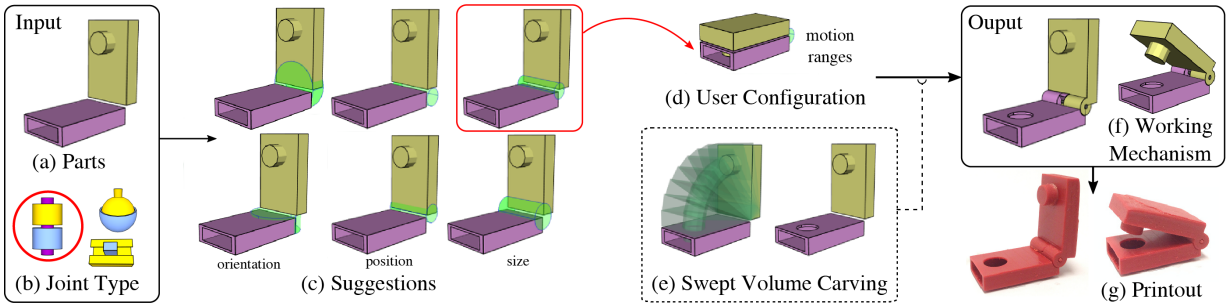


Figure 2.3: User workflow.

Connecting parts. In our system, a part consists of one or multiple combined surfaces that behave as a rigid body. Parts can be connected to each other pairwise. When two parts are connected by a joint, the system automatically adds the needed bones to the skeleton and attaches the parts to it, defining their relative movement. Thanks to the incrementally constructed skeleton, the user can immediately assess the behavior of the mechanism.

Suggestions. For a given kinematic pair, our system automatically computes a set of joint configurations (Figure 2.3c). Each configuration includes options for position, orientation, and dimensions that control the kinematic behavior and appearance of the joint. Using our interface, the user can iterate over all the configuration options and choose the one that best

fits his intended design. When the user changes an option, our system automatically updates the joint and skeleton.

Our interface displays a joint using a simplified version of its geometry (i.e. a *proxy*), which provides a visual representation of position, orientation, and size. We find that using a proxy of the geometry, instead of the full geometry, is better for depicting the kinematic behavior of the joint. For example, for a hinge, we use a cylinder that effectively shows rotation axis, position, radius, and length (see Figure 2.4a,c).

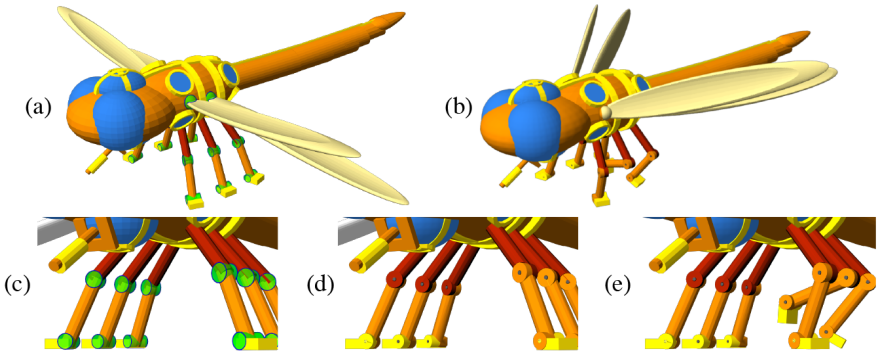


Figure 2.4: **Joint Proxies.** (a) Dragonfly model with joint proxies in green. (b) Model with actual joint geometry and posed joints. (c-e) Close-up views of the legs. Original model "Dragonfly fighter" by Alexis Zephyrian.

Controlling motion ranges. Our interface offers the user an option to specify the bounds of motion and visualize the related swept volume (Figure 2.3e-left). This visualization helps the user decide suitable bounds for the intended mechanism behavior.

After setting the motion bounds, the user can use our system to remove any intersecting geometry (Section 2.7). This step is optional and serves as an additional tool to our system: a user can decide to execute it at any point during the modeling session and as many times as required. With this tool, the user can easily carve out complex geometries from connected parts. For example, the lock mechanism in Figure 2.3 needs a hole in one of its parts to fit the cylinder of the lock. However, the size of the hole is hard to calculate: the cylinder

sweeps the volume of a torus with a minor radius equal to the radius of the cylinder, and the major radius depends on the position of the hinge.

As we show in our user studies (Section 2.8), calculating bounds and swept volumes manually is time-consuming and requires non-trivial modeling proficiency. In contrast, with our tool, even a novice user can create working joints.

2.6 Algorithms

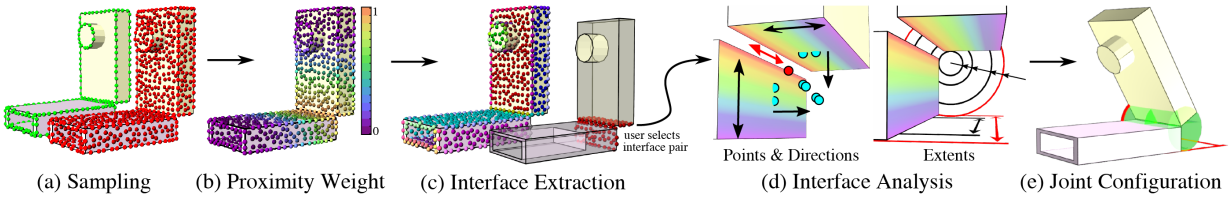


Figure 2.5: **Pipeline overview.** (a) sharp features samples (green) and surface samples (red); (b) colored proximity weight; (c) partition of interfaces, each with a unique color, and example of an interface pair selected by the user (in red); (d) points, directions, and extents calculated with our algorithm; (e) joint configuration given the point, direction and extents highlighted in red.

Our workflow for modeling mechanisms is supported by a set of algorithms that calculate the configuration alternatives offered to the user. These algorithms, with minor variations, are the same for all joint types.

The core of our suggestion algorithm is based on three observations. First, joints are often located in regions proximal to both parts (the parts's *interfaces*). Second, the motion defined by the joint is usually aligned to features of the interface. And third, the overall size of the joint must fit the space defined by these regions. All of these observations are somewhat ambiguous, and typically a number of possibilities exist, with the best choice related to the semantics of the object.

An interface is a region of a part where other components can be connected. In the case of

polyhedral parts, it is natural to use polyhedron facets as our elementary unit for interfaces. Following the three observations mentioned above, we can find interface candidates, for example, by identifying faces on both parts that are close to each other, setting the joint orientation with respect to the normals of the facets, and deciding the joint size from the face size. This approach can easily be generalized to parts consisting of smooth patches bounded by sharp curves. However, this approach works poorly for all but the simplest shapes: elements like small details, protruding sharp parts, rounded edges, curvature variation, and position or normal noise all make this simple approach unusable.

Our concept of interfaces generalizes facets. We define an interface as a set of points sampled from mesh triangles and "sharp" edges (cf. [31]). Each triangle sample is assigned a normal, and each edge sample a direction. *Intuitively, the interface is a face-like connected subset of the surface of a part, that has a boundary defined by rapid changes of normal direction, and is close to the other part.* The choice of interfaces is described more precisely below.

Pipeline overview. The input to our suggestion algorithm is a pair of parts and a joint type. The algorithm proceeds in several steps:

- The surface and sharp features of each part are sampled.
- Sample sets are partitioned into interface regions (possibly multiple regions per part, as different areas in a pair of parts may be in proximity).
- For a given interface pair, a set of attributes (positions, orientation axes, and extents) that characterize the region is computed.
- Using these attributes, a set of joint configuration alternatives is computed and scored to determine the order of presentation to the user.

Sampling. As input meshes may have highly nonuniform triangle size and shape, we start by resampling the surface uniformly, with additional samples on sharp-edges. Our sampling method is based on the Poisson disk algorithm proposed by [19] and is extended to edges. This algorithm first computes a large set, S_i, of random points uniformly distributed on the surface. Then, it draws Poisson disk samples from S_i, generating the set $S \subset S_i$. We refer the reader to [19] for implementation details. Here we only state that we can efficiently draw a Poisson disk sample set S from a sufficiently large S_i.

To compute S_i for a part p, our algorithm first triangulates the mesh and extracts sharp-edges (edges that have sharp dihedral angles or lie on the boundary [31]). Then it generates two sets of points, S_i^T and S_i^E, drawn from triangles and edges respectively. To generate S_i^T (resp. S_i^E) we repeatedly select a triangle (resp. edge) with probability proportional to its area (resp. length) and then uniformly sample the selected element. For a triangle, the sample point is defined in barycentric coordinates as $u = 1 - \sqrt{\xi_1}, v = \xi_2\sqrt{\xi_1}$, where ξ_1, ξ_2 are two uniform random numbers. Likewise for edges, the point is given by $u = \xi_1$. The union set $S_i = S_i^T \cup S_i^E$ is passed to the Poisson disk algorithm to compute the set S (see Figure 2.5a).

In addition to its position, each point is assigned a normal direction; for a point drawn from a triangle, we use the surface normal as normal direction; for a point drawn from a sharp-edge we generate two samples per point, each with a different normal (corresponding to the normals of the triangles associated with the sharp-edge). Each edge-point is also assigned an edge-direction, corresponding to the vector defined by the two vertexes of the edge.

Although we sample each part p individually, we use the same Poisson disk radius, so the densities of both sample sets are equivalent. Let a^A, a^B be the surface areas of each part, and N a desired upper bound on the size of the sample sets; then, the sample radius r is given by $r = \sqrt{(a^A + a^B)/(N\pi)}$.

2.6.1 Interface Extraction

We extract interfaces from a sample set S one by one, using a breadth-first growth for each (see Figure 2.6). To favor expansion towards regions close the other part, each point is assigned a weight using the proximity weight function defined below. The first interface is initialized with the sample point with highest weight. Starting points for following interfaces are chosen to be the points with highest weight, not yet assigned to any existing interface. Our algorithm iteratively expands the interface to neighboring points that keep the interface relatively flat.

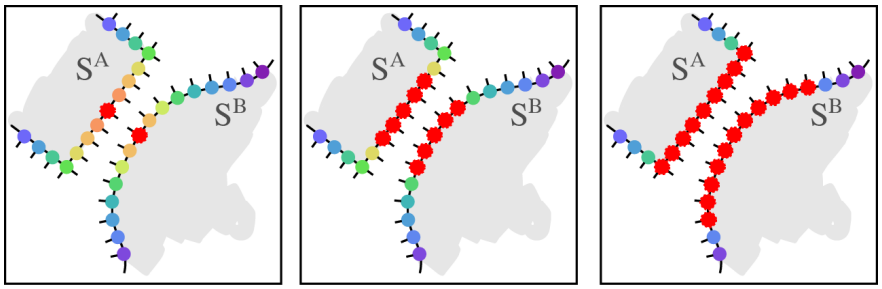


Figure 2.6: **Interface extraction.** From left to right: iterations 1, 3, and 6 of BFS in each sample set.

Let us denote the set of points that compose the interface in the k-th iteration by I_k. We define the set of "neighboring points" of that iteration as all the points in $S \setminus I_k$ at a distance less than $2r$ from any point in I_k. To keep the interface planar, our algorithm accepts a point p only if its normal direction is close to the normal of the best-fit-plane defined by I_k. We calculate this normal as the weighted average normal of the points in I_k.

The extraction of interface I ends when no neighboring points are found that satisfy the plane requirement. The algorithm continues extracting new interfaces until all points have been processed and the whole surface is segmented.

Proximity weights. Motivated by the observation that joints are typically located in regions close to both parts, we weight points of a sample set by their distance to the other

set. Due to their good localization properties, we choose these weights to have a similar form as weights used in moving-least-squares constructions. For a point $p \in S^A$, its weight with respect to S^B is given by:

$$w_d(p, S^A, S^B) = \exp(-c_1 \|p - q(p, S^B)\|^2 / a_{min}), \tag{2.1}$$

$$a_{min} = \pi r^2 \min\{|S^A|, |S^B|\} \tag{2.2}$$

where $q(p, S)$ corresponds to the closest point on set S, and r is the sampling radius. The parameter c_1 controls the falloff of the weight function. All the examples in this chapter use $c_1 = 100$.

To accelerate the subsequent stages, we remove most of the resulting interfaces for each pair of parts from consideration, keeping only the ones that have significant weight and size: given an interface set $I \in S$ we use a threshold on the total weight of its points (10% of the total weight of points in S) and a size threshold (1% of the area of S).

2.6.2 Computing Positions, Axes and Extents

For a pair of interfaces, our algorithm computes a set of positions, axes, and extents that are used to configure the parameters of the joint (see schematics in Figure 2.7). The sets of points, axes, and extents we use are based on observations of a gallery of joints used in current manufacturing (see examples in Figure 2.8).

Positions. A position is a possible location of the center of a joint. We extract two types of positions, *surface* points and *bisecting* points, according to their distance to the interface planes. Surface points lie on the plane defined by the interface and are used to position the joint inside a part (see the slider joint in Figure 2.8a). Bisecting points are equidistant to both interfaces and are particularly relevant to symmetric joint models (see hinge joint in

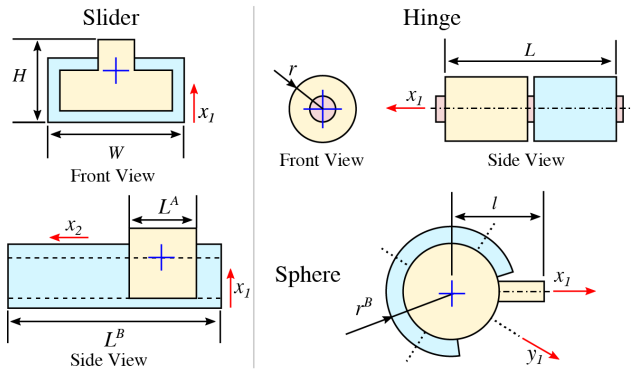


Figure 2.7: Schematics for joint models. Our system automatically calculates values for the position (indicated by a blue cross), part orientations (red arrows), and part dimensions. Dimensions not included in this figure are derived from these main dimensions. Such is the case of the radius of the ball in the sphere joint, calculated as a fraction of r^B. Complete schematics of the models are included in Section 2.10.

Figure 2.8d).

We compute the following alternatives for positions (Figure 2.9):

- Points p_0 and p_1 lie on the interface and correspond to its unweighted and weighted center of mass respectively (eg. Figure 2.8a,c,f,h).
- Points p_2 and p_3 lie in the bisecting plane and correspond to the intersection of the plane and the line defined by p_0 (resp. p_1) and the interface normal (eg. Figure 2.8e,g).
- Point c corresponds to the weighted center of mass of both interfaces and usually lies on the bisecting plane (eg. Figure 2.8b,d).

Axes. An axis is used to orient a natural body-aligned axis of a joint (e.g., the rotation axis of a hinge, or the slider direction). An axis can be *tangent* or *perpendicular* to the interface average plane. For example, in the slider model in Figure 2.7, the axis x_1 is perpendicular to the interface while x_2 is tangent.

We use the following directions for a given interface pair:

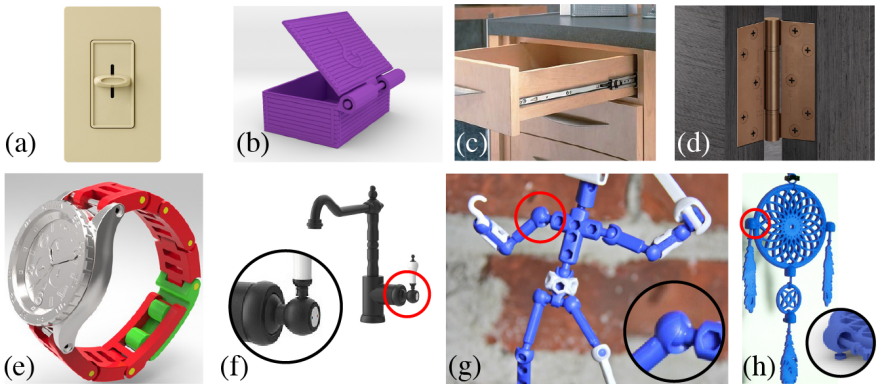


Figure 2.8: Gallery of mechanisms from online repositories: (a) light switch with slider joint, (b) box with hinge joint, (c) drawer with sliding joint, (d) door hinge joint, (e) watch band with multiple hinge joints, (f) faucet with sphere joint, (g) toy, ModiBot, with multiple sphere joints, and (h) dream-catcher with sphere joints.

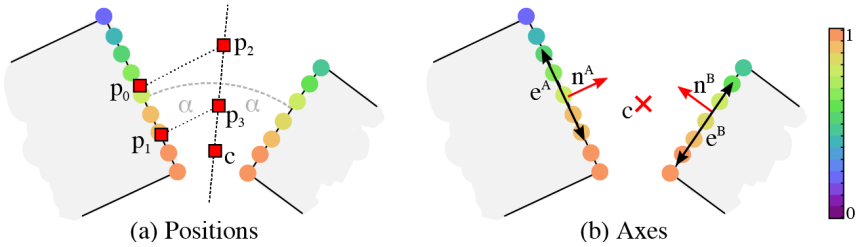


Figure 2.9: Positions and axes calculated by our system.

- Best-fit-plane normal of each interface: for axes perpendicular to an interface (e.g., sphere joints on Figure 2.8f,g, and axis x_1 of slider joints on Figure 2.8a,c));
- Cross product between best-fit-plane normals: for axes tangent to both interfaces. This is particularly useful when the initial pose is already rotated, as it indicates the preferred rotational axis (e.g., Figure 2.8 b,d,e,g).
- Sharp-edge directions: for axes tangent to one of the interfaces (e.g., travel axis x_2 for slider joints on Figure 2.8a,c).

To compute salient sharp-edge directions, we use histogram-based clusters of directions. Using the edge direction, our method projects all sample edge points to a 3D cube grid. It then picks two cells (bins) with the highest total weights. When no relevant bins are found,

it uses PCA to compute two principal directions for the interface. We find this approach adequate, since we need only a relatively coarse resolution.

Extents. An extent is a range of distances given by a lower and upper bound. We use the following types of extents (see Figure 2.10): linear, radial and spherical radial. For example, models for sliding joints include mostly linear extents (e.g., Figure 2.8a,c); hinges include linear and radial extents (Figure 2.8b,d,e); and ball-and-sockets include mostly spherical radial extents (e.g., Figure 2.8f,g,h).

A *linear* extent is associated with an axis a and a position p, and is obtained by projecting all interface points to a line l in direction a passing through p and measuring the distances to p.

A *radial* extent is also associated with an (axis, position) pair, but the distances from sample points to the line l are measured.

A *spherical radial* extent is obtained from distances from sample points to a position p.

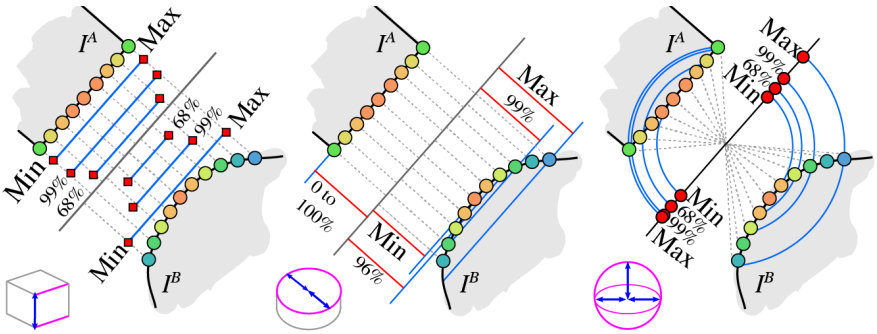


Figure 2.10: **Extents Measure.** Interface I^A represents an interface with a sharp bound, and I^B one with soft bounds. For a given extent, distances are measured (a) along a line, (b) to a line, or (c) to a point, and used to obtain representative values for the extent.

For each set of distances, we compute the ranges that cover the 68%, 95% and 99% of the interface total weight, with distances weighted by the weight of corresponding points. We also obtain the extreme points, min/max, along the line.

Interfaces in which weights are similar (see I^A in Figure 2.10), have very similar 99% and maximum ranges. In contrast, interfaces where the weights decay slowly (e.g., I^B) present a significant difference between the 99% range and the extreme options. In the first case, the extreme range often offers a better solution, working as a "snap to border" option. In the second case, the extreme range is not representative of the interface (having a low weight). To offer appropriate solutions to the user, our algorithm evaluates the difference between the 99% range and the extreme range: if the difference is smaller than twice the sampling radius, only the extreme options are offered. Otherwise, the percentage options are used.

The values calculated are used to modify the parameters of the joint's model (Figure 2.7). In most cases, the value of the parameter is calculated directly from a single extent. However, some parameters are affected by multiple extents. Such is the case of the number of knuckles in the hinge joint, as shown in Figure 2.11. Here the number of knuckles is an integer, calculated as a relationship between the length and radius.We show the complete parametric models and the relationships between the attributes in the supplementary material.

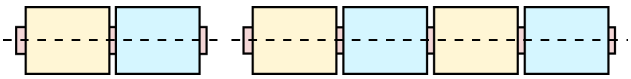


Figure 2.11: **Indirect parameters.** In the hinge model, the number of knuckles is calculated as a factor of the ratio between the length and the radius of the joint.

To add new models into our system, it is sufficient to specify how to calculate the parameters in terms of our main dimensions (see Figure 2.7). For models with different main dimensions, each dimension needs only to be classified as linear, radial, or spherical radial along a pair of position/axes.

2.6.3 Computing Configuration Alternatives

When the user selects a pair of parts, our system automatically calculates a set of interface pairs (Section 2.6.1) and alternatives for positioning, orienting and sizing the joint

(Section 2.6.2). These alternatives are intuitive to the user, as they have a direct relation to the common 3D manipulation tasks of translating, rotating and scaling. Before the options are presented to the user, each set of alternatives is sorted and the option with highest score is used as default. While position and axis choices are only dependent on the interface pair, the extents depend on both and so they need to be computed separately for each (axis,position) combination.

Interface pairs are ranked by the proximity between their points. For a pair (I^A, I^B), the weight is given by:

$$w_I(I^A, I^B) = \frac{1}{|I^A||I^B|} \sum_{p \in I^A} w_d(p, I^A, I^B) \sum_{p \in I^B} w_d(p, I^B, I^A) \tag{2.3}$$

where w_d is defined as in Equation 2.1.

Similarly, position alternatives are ranked by their proximity to both interfaces, with the weight of a position p given by:

$$w_p(p, I^A, I^B) = w_d(p, I^A, I^B) w_d(p, I^B, I^A), \tag{2.4}$$

Positions originating from different interfaces may be close; we consider a position p to be a duplicate if there exists a position q such that the distance between them is less than a tenth of the sampling radius. Only the position with the highest weight is retained.

Axis alternatives of different types are sorted differently: for tangent axes, we offer first the cross product (if any) and then the sharp edge directions; for perpendicular axes, we offer the normal directions. Axes with angular difference of less than 0.1 radians are viewed as identical.

Extent alternatives are sorted by length, from shortest to longest. We consider a extent $[t_1, t_2]$ to be a duplicate if there is another extent $[l_1, l_2]$ whose difference between corresponding

endpoints is less than a tenth of the sampling radius. Duplicates are handled by choosing arbitrarily between the options.

2.7 Generating Physically Realizable Geometry

Once the user is satisfied with the configuration of the joint, our system creates its geometry and connects it to the mechanism. This involves three steps: generating working joint geometry, adding gaps between the parts, and computing the swept volume to ensure the parts can move.

Generating working and printable geometry. Before connecting the geometry to the mechanism, it is necessary to remove any volume that interferes with the pair's relative motion (see Figure 2.12). Let us denote the joint geometry that will be attached to part A as J^A (resp. J^B for part B). To connect J_A to A, our algorithm first computes the volume V^B traversed by J_B (see below) and carves it off A using Boolean difference. Finally, J_A is connected using Boolean sum. The same operation is done for B, carving off V_A and adding J_B, with V_A computed for the original J_A. Our algorithm adds a separation between the parts (clearance) by inflating V_A and V_B before carving. An appropriate clearance value depends on the printer's precision and is therefore provided by the user.

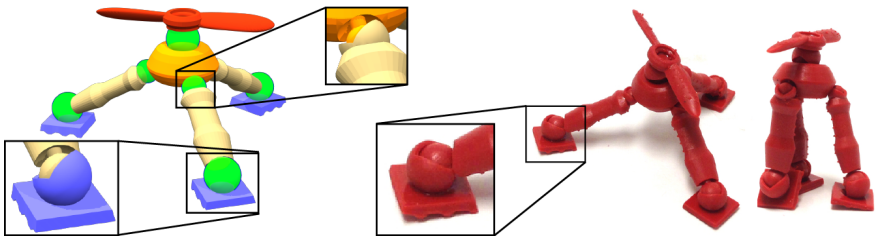


Figure 2.12: **Printable Geometry.** Propeller Head model. Geometry is carved out of the parts to allow joint motion.

Given the arbitrary shape of the parts, some remaining geometry, outside the vicinity

of the joint, might still interfere with the part's movement (e.g., the highlighted cylinder in Figure 2.13). Because our tool is meant to be used as part of the modeling process we do not automatically remove this geometry, but allow the user to run a sweep analysis at any time during modeling and selectively remove unneeded regions.

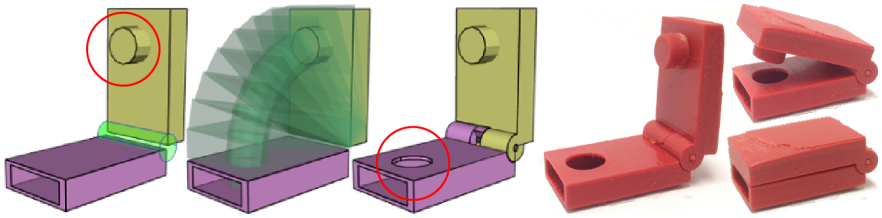


Figure 2.13: Lock model. Our system automatically carves the purple part to fit the cylinder on the yellow part.

Swept Volume Computation. Several generic algorithms exist to obtain a swept volume generated by arbitrary transformations of a mesh over time. We found that the existing code available to us to compute swept volumes was not robust and was therefore unsuitable for our system. Thus, we implemented our own simplified algorithm similar to the method described in Peternell et al. [72]. Their method finds characteristic or silhouette points in the direction of motion at each time step and sweeps them in that direction at the step intervals, creating a mesh from this set of boundary points. We tailor this method for the simpler, constant-velocity trajectories which mechanical objects typically create, e.g. lines, circular arcs, and helixes. As in most swept volume algorithms, the most computationally demanding aspect of our approach is the use of robust Boolean operations [104] to identify the inside and outside of the shapes.

We explain our approach using the case of linear one-dimension translation. The algorithm first partitions the mesh of the moving part along the silhouette in the direction of the translational sweep, separating it into regions of front-facing and back-facing components (components whose vertex normals have a positive or negative dot product with the direction

of the sweep). Then, for each separate patch of connected front-facing or back-facing components, we construct the swept volume: two copies of the patch are created and then translated respectively to the starting and ending position of the sweep, and the corresponding boundary vertices are connected. If we take the Boolean union of all patch sweeps and the original shape, we can create the entire swept volume. However, for efficiency, we consider only those patches whose sweeps intersect with the static part. Note that if the starting shape is manifold, then this method, by construction, produces closed manifold meshes.

This method can be used for other types of one-dimensional parameterized motion using a simple change in coordinate system. For example, for the hinge, we remap the circular rotation to a simple linear translation, mapping the Cartesian coordinates to polar coordinates relative to the rotation axis. The sweep is created using the same process described above, with the addition that, in order to create a smooth curve, multiple vertices are added to interpolate between the boundary vertices of each pair of swept patches. Robust boolean and mesh operations are used to find the union of the patches, resolve self-intersections, and compute an outer hull of the resulting mesh. For more complex types of motion with multiple independent degrees of freedom, such as the motion allowed by a ball-and-socket joint, the sweep must be computed successively for each degree of freedom: the shape is first swept around one axis, the resulting shape is swept around the next axis, and that result is swept along the final axis.

2.8 Discussion and Results

To evaluate our system, we conducted pilot experiments with novice users and applied our system to create mechanical objects of varying complexity. In order to facilitate the evaluation, we implemented our system as a plug-in for Maya 2015 — which we will release online to support and foster research in this area.

2.8.1 Pilot Experiments

We conducted the experiments on a MacBook Pro equipped with a 15-inch screen. The test group consisted of eight subjects: four Computer Science students and four architects. None of them had extensive experience with graphics modeling software. Half of them reported having low to no experience with 3D modeling and rigging.

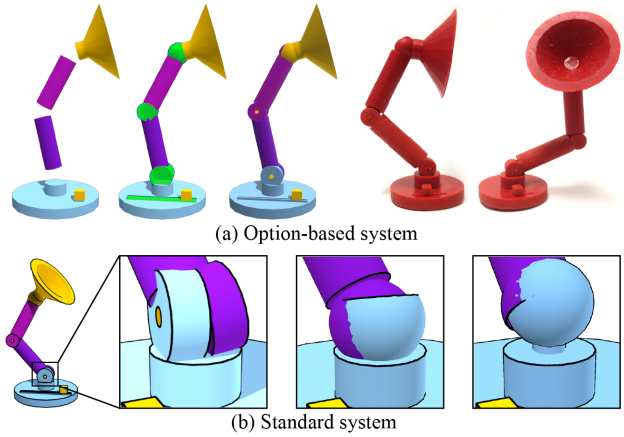


Figure 2.14: **Task I.** Lamp model (a) Result using our option-based system. (b) Results using traditional modeling tools.

Task I. To validate our suggestion-based approach, we asked users to connect the Lamp model (Figure 2.14) using our system and to repeat the task using standard modeling tools present in Maya. For the standard workflow, we provided participants with parametric models of the joint's geometry.

When using our system, the user would choose one of the offered suggestions for position, orientation and dimensions of the joints, with the possibility of minor adjustments of joint parameters at the end. In the standard modeling trial, the user was free to use standard tools for translating, rotating and sizing the parametric model and skeleton. In both cases, participants were allowed to change the camera viewpoint. The trial was considered completed when the user determined that the model and skeleton of each joint "looked correct." Before

the test, participants were briefed on the kinematics of the joints, and they practiced with both systems until they felt comfortable with the interface.

To compare the performance of the two systems, we measure the total completion time and the edit time. Edit time considers the time spent modifying the configuration of the joint: choosing one of the alternatives in our system, and translating, rotating and scaling in the standard system. Total completion time includes the additional time spent manipulating the viewpoint. For usability comparison, we use the System Usability Scale (SUS) questionnaire [12].

Manipulation	**Standard**	**Option based**	**Speedup**
Translation	185.1s	36.4s	5.1 x
Rotation	161.3s	25.5s	6.3 x
Scaling	138.5s	85.1s	1.6 x
Camera	613.3s	196.5s	3.1 x
Edit Time	484.9s	146.9s	3.3 x
Total Time	1099.3s	384.0s	2.9 x

Table 2.1: Average performance on Task 1, for standard and option-based systems.

Results. The results show that with our system users took considerable less time to finish the task, with a speedup of 3.3 times on edit time and 2.9 times on total time (see Table 2.1). The improvement derives not only from the time spent directly configuring the joint but also from the time spent manipulating the camera (speedup of 3.1 x). Camera changes during the standard trial were common when the user switched between editing the joint and checking the overall aspect of the mechanism. For example, the user would zoom in to change the length of the hinge and then zoom out and orbit to see how it looked. This behavior was frequent in the standard trial and subdued in the option-based trial. We conjecture this is due to needing few different viewpoints to evaluate our options. Additionally, changes of previous decisions were more common for the standard system, most likely due to errors discovered while inspecting the joint from a new viewpoint. This data supports our belief

that the standard pipeline needs multiple iterations to solve issues, while our option-based approach solves them in fewer steps. Furthermore, it is important to point out that most of the resulting joints modeled with the standard system had mismatching geometry and skeleton (see Figure 2.14b), which could cause errors during animation (parts separate or intersect) and unexpected motion or collisions in fabricated models.

On the usability questionnaire, the option-based system has an average score of 86.9 pts on a scale of 0 to 100 pts. According to Bangor et al. [12], this score translates to an adjective rating between Excellent and Best Imaginable (B in grade scale). In contrast, the standard system, with an average of 37.8 pts, is qualified as Poor (F in grade scale).

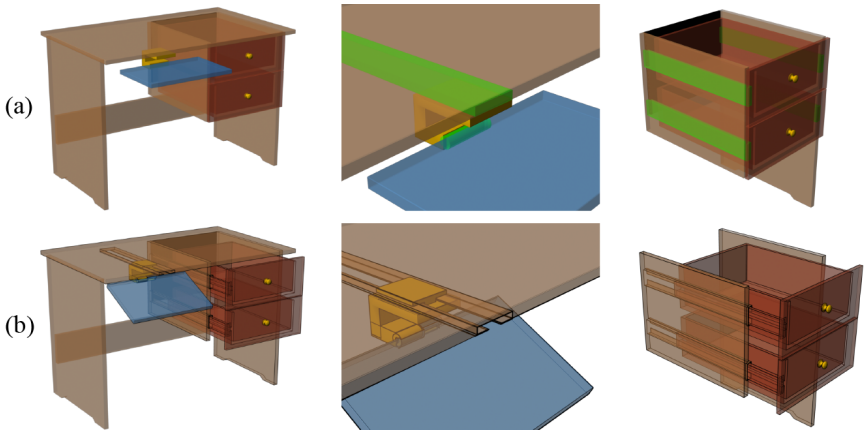


Figure 2.15: **Task II.** (a) Desk model and close-up with joint proxies; (b) Results obtained with our option-based system. Model derived from "Wooden Desk" by Carlos Folch.

Task II. This was an informal study to retrieve more feedback on the perceived accuracy of our options. Participants were asked to add joints to the Desk model (Figure 2.15) using our option-based system.

Results. The feedback from the users indicated that our system offers good solutions for position, orientation and size. All subjects agreed that they found alternatives similar to what they wanted and that the final model looks close to what they had envisioned. They

also agreed that most of the solutions offered made sense and that they were likely to check all possible options.

2.8.2 Mechanical Objects

Figures 2.1,2.4,2.12-2.19 show a range of mechanical objects of varying complexity with joints modeled using our system. Some included a large number of joints and long chains (e.g., Figure 2.1 and 2.16a), others contained faces with noise, small details, or intersections (e.g., Figures 2.4,2.16b,c), and others had relatively simple geometry with few joints (e.g., Figures 2.14a,2.15,2.19). The models in Figures 2.12,2.13,2.14,2.17,2.18,2.19 were created from scratch by novice users with inspiration from real-life mechanisms. The rest were obtained from online stores, like Shapeways and Turbosquid.

For each object, we used our system to generate, adjust, and implement joint geometry. In cases where the model included non-operational joints (Figure 2.16), we removed that geometry beforehand. Using our system, we were able to create joints that look similar to the original design. For the Retro Robot model, we manually adjusted the shoulders joint (moved the joints up, and extend their length). For models with loops (Figures 2.17,2.18), it was necessary to break the rig, as Maya's rig system doesn't support loops.

To demonstrate that the results are physically realizable models, we fabricated six of the mechanisms using a B9Creator DLP/SLA printer with Cherry Red resin, and post-assembled the models. The printing process tends to cause inflation, and although our models included a clearance between the parts, the inflation eliminated it. Therefore, the fabricated parts are in contact, creating friction. To control the degree of friction, we altered the models' clearance values: 0.2mm (high friction) to 0.4mm (low friction). For pre-assembly printing, other techniques (eg.[22]) can be used to provide friction. Incorporating these additional joint models is straightforward: the same parameters that configure our ball-and-socket are used

(a) Retro Robot (b) Chameleon (c) Dalek

Figure 2.16: Models with non-operational joints, which are shown in yellow in the first image of each set. Using our system, we were able to create similar-looking joints. Original models (a) "Retro Robot 1" by Forpost D6, (b) "Dino-Robot" by Ms. McClure, and (c) "Dalek" by squidinc3d.

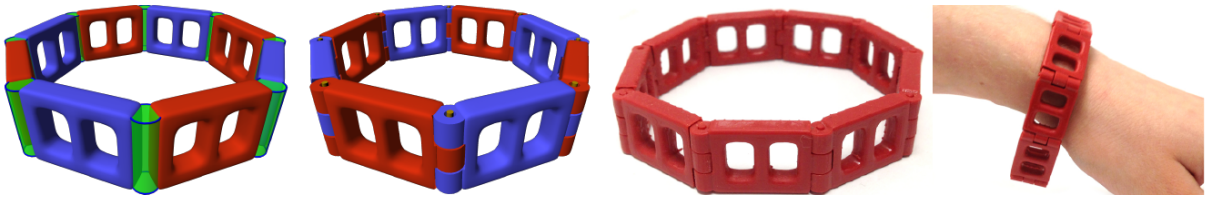


Figure 2.17: Bracelet model. (left) Model with joint proxies; (center) Model with joint geometry; (right) Fabricated mechanisms.

to configure the cage-and-socket (see Figure 2.20).

Limitations We restricted our implementation to three joint types. However, the same approach can be applied to other kinematic pairs or even higher pairs such as gears.

The major limitation of our system is that joint configurations are calculated using attributes of only two interfaces. While this is sufficient for many joints, artists sometimes want to match *global* attributes of the mechanism. For example, in the Dalek model, an artist might want to force all joints to have the same radius. Our current system can certainly be extended to handle attributes extracted from other regions, but this would require new shape analysis methods.

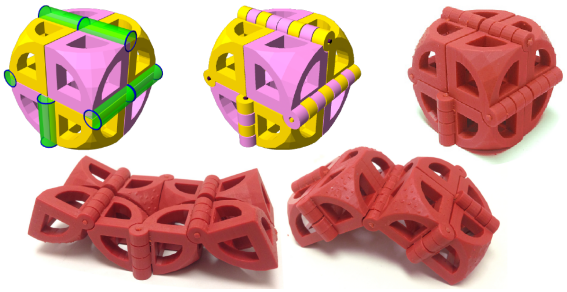


Figure 2.18: Magic Cube model. (left) Model with joint proxies; (center) Model with joint geometry; (right-bottom) Fabricated mechanisms.

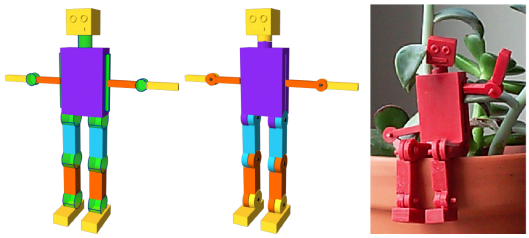


Figure 2.19: Tiny Robot model. (left) Model with joint proxies; (center) Model with joint geometry; (right) Fabricated mechanisms.

The heuristics built into the system are only as powerful as the detected interfaces. To properly detect features, a suitable sampling radius r must be used: details smaller than r are overlooked, while larger noise might cause an interface to separate. Also, since we sample only the surface, we might fail to detect proper interfaces on parts with large intersection volumes. In such cases, user intervention might be required to modify or reposition the parts. Finally, our algorithm requires parts to be placed in an initial pose that is allowed by the desired relative motion. Integrating our system with tools that assist in this task (e.g., [8], Tinkercad, Gravity Sketch) is left as future work.

Figure 2.20: Lamp model printed using joint model from Calì et al. [22].

Last but not least, we do not take into account physical properties of produced models, only kinematics. Accounting for physical effects is important for fabrication applications. One approach for supporting physical effects would be to combine our method with stability and strength evaluation [75, 106, 11, 60] to find, for example, structurally optimal choices in the parametric space of joint positions and dimensions.

2.9 Conclusion

We have presented a system that allows inexperienced users to create articulated, physically correct objects easily. The basic approach of our system, presenting a small set of options to the user for each operation, proved to work quite well, either providing an immediate satisfactory answer or a close approximation point which could, with ease, be adjusted to get the final shape. Our system is based on a set of carefully selected heuristics that do not require manual or automatic model segmentation, except for partitioning the model into moving parts. The underlying computations are fairly simple and do not require an extensive database of sample objects or complex crowdsourcing and learning, yet they yield, in all cases we have tried, small sets of choices including several semantically natural ones.

2.10 Chapter Notes

2.10.1 Supplementary Material: Joint Schematics

This section describes the parametric models for all joints used in our chapter. We group the parameters into three sets:

- **Direct parameters.** Parameters whose values are calculated using our option-based system.

- **Indirect parameters.** Parameters whose values are derived from the direct parameters.

- **User-defined parameters.** Parameters set by the user through our interface.

To prevent fusion during printing, we separate different parts of the geometry by a distance c (clearance). This parameter, which is common across all joint types, depends on the printer and is therefore provided by the user.

Hinge A hinge implements a one-dimensional rotation kinematics defined by an axis of rotation (a line in space given by a direction r_a and position p) and the minimal and maximal rotation angles about the axis.

As a prototype, we use the most common mechanical design, a "barrel hinge," which is a sectional barrel secured by a pin (Figure 2.21). The barrel is composed of n pipe-shaped sections, called knuckles, that provide the rotational axis of the hinge. Each section is attached to only one part of the kinematic pair: even sections are attached to part A while odd sections are attached to part B. Along the length of the barrel, knuckles are separated from each other by the clearance distance c.

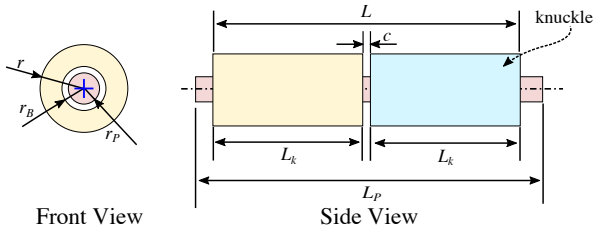


Figure 2.21: Geometric parameters of the hinge joint.

The rotation of the hinge is limited using interlocking notches in the knuckles (Figure 2.22). By default, our system creates a hinge with no limits, and the user must set the forward and backward angles (θ_1 and θ_0) using our interface. The length of the notches, l_k, is derived from the length of the knuckles (Table 2.2), but can be overridden by the user.

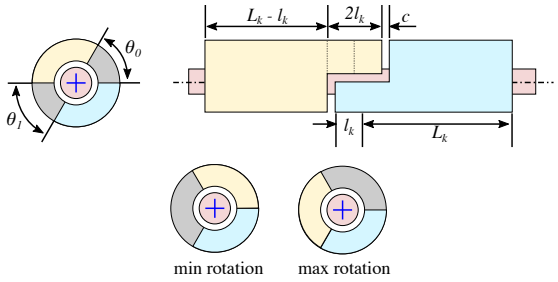


Figure 2.22: Rotation limits of the hinge joint.

Direct Parameters		
r	barrel external radius	
L	hinge total length	
Indirect parameters		
L_P	pin length	$L_P = L + 2c$
r_P	pin radius	$r_P = 0.2r$
r_B	barrel internal radius	$r_B = r_P + c$
n	number of knuckles	$n = \lceil L/(2r) \rceil$
L_k	length of each knuckle	$L_k = (L - c(n-1))/n$
l_k	length of rotation constraints	$l_k = 0.1L_k$
User-specified parameters		
θ_0	minimum angle of rotation	
θ_1	maximum angle of rotation	

Table 2.2: Description of geometric parameters of the hinge joint.

Sphere A spherical joint allows an arbitrary rotation about a single point, the joint center, within specified bounds.

We use a common type of joint consisting of a spherical socket, which is directly attached to part A, and a ball with a pin, which is attached to part B. The ball and socket are separated from each other using the clearance distance c. The socket thickness d is set by default as two times the clearance but can be overridden by the user.

Slider The slider allows for a single translational degree of freedom along a common line. Since many different cross-sections can allow this type of motion, slider joints can have many different geometries.

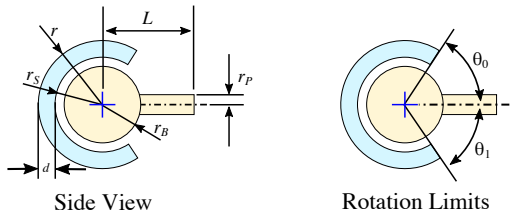


Figure 2.23: Geometric parameters of the ball-and-socket joint.

Direct Parameters		
r	socket external radius	
L	pin length	
d	socket thickness	default $d = 2c$
Indirect parameters		
r_B	ball radius	$r_1 = r - t$
r_S	socket internal radius	$r_0 = r_1 + c$
r_P	pin radius	$r_p = 0.1r$
User specified parameters		
θ_0	minimum angle of rotation	
θ_1	maximum angle of rotation	

Table 2.3: Description of geometric parameters of the ball-and-socket joint.

Our basic model for a slider is a "rail" slider similar to the type used in furniture. This model consists of two interlocked parts, the rail and the slider. The slider, shown in yellow in Figure 2.24, has a T-shape that prevents its translation along other axes. The rail is a hollow box whose thickness d is set by default as twice the clearance. The length of the rail, L, defines the translational limits of the slider. Its value is set directly by our system and, similar to the rotational constraints for the hinge and sphere, can be manually overridden by the user.

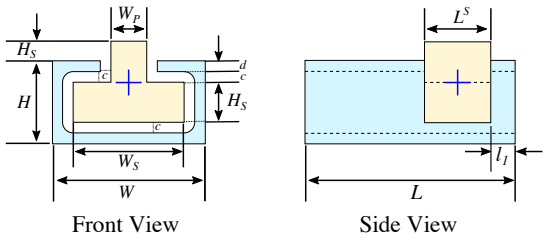


Figure 2.24: Geometric parameters of the slider joint.

Direct Parameters		
W	rail width	
L	rail length	
L_s	slider length	
H	rail height	
H_s	slider height	
d	rail thickness	default $d = 2c$
l_1	initial translation	

Indirect parameters		
W_p	pin width	$W_P = 0.33 * W$
W_s	slider width	$W_S = W - 2(c + d)$
H_s	slider height	$H_S = H - 2(c + d)$

Table 2.4: Description of geometric parameters of the slider joint.

Chapter 3

Reinforcement of Shell Structures

This chapter is based on the publication Gil-Ureta et al. [32] in collaboration with Nico Pietroni and Denis Zorin. For this project, my contributions include development of continua optimization, which is a generalization of Hencky-Prandtl, and material distribution optimization; performance analysis of our method; comparisons with previous work; and designed and execution of physicall experiments.

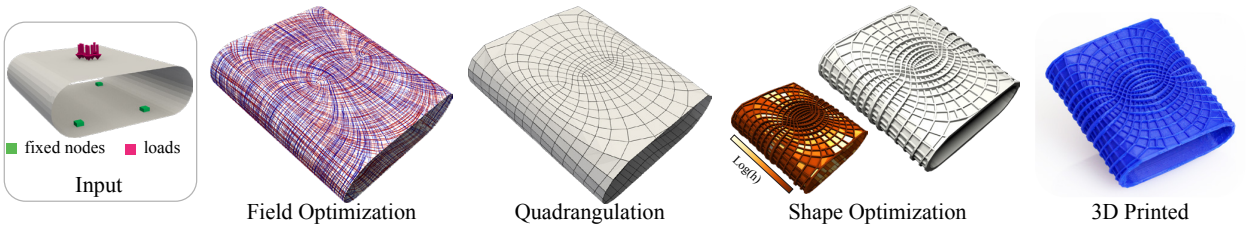


Figure 3.1: Stages of our pipeline for generating a shell reinforcement structure.

3.1 Introduction

In structural design, shells are considered to be one of the most efficient structures because they can be simultaneously lightweight and robust. Shells are common in additive fabrication applications because using shells instead of solids reduces the cost of material and decreases

the fabrication time.

An optimally-shaped shell can carry its load relying only on tensile/compression forces, with no bending involved, which is very efficient in terms of the required material. These types of shells are commonly found in architecture (domes). However, the shape of the shell may be determined by considerations other than its load-carrying properties. For example, the shape of the airplane is determined by its aerodynamics; the shape of the car body both by the aerodynamics and aesthetics; the top of a table or a shelf is flat, as objects need to be placed on it; the artistic intent primarily determines the shape of a lamp or a statue.

Shell structures with shapes fixed by considerations other than loading are often reinforced by additional means, most commonly increasing thickness in critical areas or adding ribs (Figure 3.2). Formally, a common optimization goal for a shell reinforcement structure is to *minimize the weight of the added material while keeping the maximal stress of the structure bounded*. The first ensures the structure remains lightweight, while the second prevents structural failure.

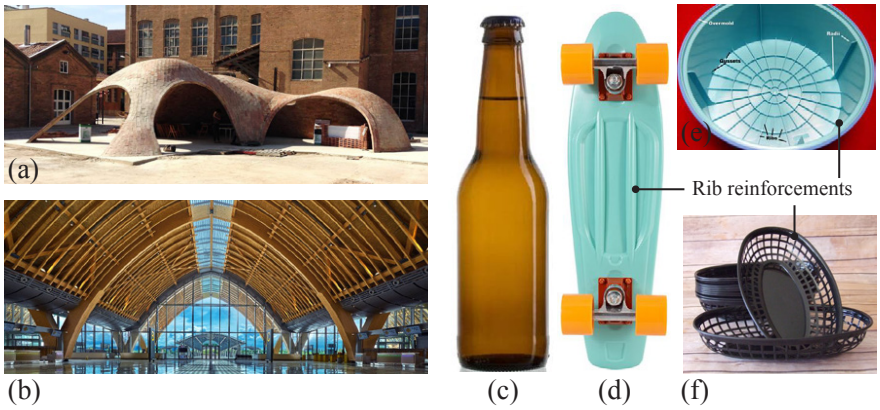


Figure 3.2: Examples of shell structures showing variable thickness walls and ribs: (a) Bricktopia Pavilion, Barcelona, has regions with one, two, or three layers of bricks; (b) Mactan Cebu International Airport, Philippines; (c) glass bottles of variable thickness; (d) plastic skateboard (1.75kg, supports 100kg); (e) plastic bucket; (f) plastic food basket.

This problem has been well studied for two dimensions in the limit of low volumes. In 2D, the optimal layout forms a pattern of orthogonal lines (Hencky-Prandtl net) and, for a given

layout, the minimum weight structure has all members fully stressed. These structures form, in the limit of low volumes, classical Michell structures and can be obtained by solving a convex optimization problem. It is also well understood for pure bending problems for plates, i.e., the special case of flat shells with loads orthogonal to the surface; in this case, it also reduces to a different convex problem.

The situation is far more complex for the reinforcement of shells embedded in three dimensions. For these shells, the weight-optimal structure may be locally either beam-like, forming *ribs* aligned with stress directions, or membrane-like, forming variable-thickness walls with no perforation [83]. The first case typically corresponds to bending-dominated regions while the second to areas dominated by in-plane forces. The optimal local structure is determined by the surface shape, the supports, and the loads.

In this general case, the problem is no longer convex and cannot be optimally solved either by methods that assume that result is only a variable thickness shell or by Michell-truss type methods.

In this chapter, we propose a novel efficient computational method for constructing optimized reinforcement structures for shells, naturally producing a full range of behaviors spanning the space between variable-thickness shell and rib-type reinforcement. Our approach can be applied to reinforcing any types of free-form manufactured objects: 3D printed plastic or metal shapes, as well as structures produced by casting.

We partition the problem into three steps (Figure 3.1): (1) determine the field of (approximately) optimal stress directions; (2) construct the skeleton of the reinforcement structure that follows these directions, forming polygonal (predominantly) quad cells aligned with the field; (3) optimize how material is distributed inside the cells.

Main contributions.

- For computing the field of optimal stress directions, we developed a generalization

of Hencky-Prandtl nets which takes bending into account and can still be solved by minimizing a convex energy.

- For material distribution optimization, we use a low-parametric structure model for cells to efficiently optimize the distribution of the material. As the global optimization problem is *fundamentally non-convex* (we discuss the reasons on Section 3.3), to solve it we propose an efficient global/local method which shows stable and fast convergence behavior.

We validate our approach by optimizing several 3D shapes. This evaluation shows that our method can handle shells with arbitrary curvature, and successfully transitions between membrane- and bending-dominated regions, obtaining the expected optimal substructures. We demonstrate that, by optimizing jointly for bending and compression/tension dominated regimes, we obtain lighter structures than previous work.

3.2 Related Work

Our work builds on the ideas from classical structure design for 2D elasticity and plates, with the key ones originating the work of Michell [63].

We complement these fundamental ideas with quadrangulation techniques which can be reinterpreted as a way to transition from an infinite continuum of field-aligned beams to its discretization. We use a variation of Bommes et al. [18], but any conforming, field-aligned method could be used (e.g., [46, 67, 3, 23, 30]). The optimization method for computing the optimal strain field can be viewed as a specialized cross-field optimization method. Similarly to the recently proposed method of Knöppel et al. [50] it has the advantage of being convex.

We refer to Vaxman et al. [96] for a complete overview of the related work on field design and to Bommes et al. [17] for quadrangulation.

Shell Optimization The closest works to ours are the recent works Kilian et al. [49] and Li et al. [57], with which we share a number of ideas. The former describes an elegant connection between curvature and Michell trusses and optimizes the surface shape so principal stress and curvature directions coincide. Only tensile forces are considered, and the volume approximation they use is valid for narrow beams (see Section 3.3). Similar to our work, Li et al. [57] keeps the shell surface fixed. This work considers a network of ribs, aligned with stress lines, and minimizes their volume; similarly to Kilian et al. [49], this work also uses a narrow-beam approximation for the volume, and always produces a thin-beam structure. The cross-section shape of individual beams is optimized, which produces additional weight reduction. We discuss differences to these works in more detail in Section 3.8.

Our approach also shares similarity with Groen and Sigmund [36], which uses similar structures for constructing 2D optimized structures.

On the other extreme, Zhao et al. [103] considers optimization of variable shell thickness, while keeping the topology fixed, which is suboptimal for bending reinforcement. Our work aims to bridge the gap between these extremes.

The approach of Pietroni et al. [73] aligns a network of beams to an input stress field. Another recent related work, Jiang et al. [45] considers structures made out of beams with a small number of distinct cross-sections. Both methods are suitable for architectural design; instead, we focus on applications, like 3D printing, which allow greater flexibility of structures.

Structural Optimization The literature on structural optimization is quite extensive, and there is no chance that we can do justice to all of it. The main types of approaches found in the literature include topology optimization methods (SIMP or ESO-based), analytic methods for optimal structures directly based on Michell-type theories, and methods based on shape derivatives (using an explicit or implicit evolving surface representation). Important books, which include reviews of many other works are Rozvany [78], Allaire [4], Bendsøe and

Sigmund [15], as well as recent reviews, Munk et al. [66] and Sigmund and Maute [84].

The most prevalent methods in topology optimization of structures are based on SIMP-type methods (see Bendsøe and Sigmund [15] for a review), which relax the problem to optimizing a density over a domain, which is then converted to a structure by thresholding. This approach has many advantages, including simplicity of implementation [82], connection to homogenization theory, flexibility in integrating functionals, and ease of scalable implementation [1, 98]. Nevertheless, the result will typically depend on initialization: for complex topology to emerge, the domain needs to be discretized using a fine grid. The parameters of the result (e.g., the sparsity of the structure, or minimal thickness) need to be controlled indirectly through algorithm parameters. Finally, the result is a voxelized structure, which then needs to be converted in some way to a form more suitable for manufacturing. In comparison, our method directly produces solutions based on a *globally optimal* field (in low-volume limit) and a beam skeleton for the optimized structure, which can be directly adjusted by the user in a variety of ways (e.g., converted to a spline-based CAD model if desired). We compare in more detail in Section 3.8.

Ground structure methods are among the oldest methods for optimizing topology of truss/beam structures. These methods start with a structure consisting of a large number of redundant beams and optimize it to determine the cross-sections, which automatically eliminates some of the beams. Recent examples of applying these type of methods include Sokół [86], and Zegard and Paulino [100, 101]. Compared to our approach, ground structure methods have to restrict the directions of beams to a small set, which affects both optimality and flexibility of the design. The larger the initial set of beams, the closer they may approximate the optimal result. In computer graphics, the ground structure method was used early in Smith et al. [85] for truss structure design. Panetta et al. [71] used a version of a ground structure method to obtain initial topologies for computing microstructures with prescribed material properties, followed by shape optimization.

To a great extent, our work was inspired by the beautiful structures explored in the literature on analytic or semi-analytic structure design, e.g. Rozvany [79], which includes many examples of exact problem solutions, such as Hencky-Prandtl nets. Our goal is to use this type of ideas in the general setting of curved surface domains, taking advantage of the optimality criteria and insights into the structure of the solutions. A concise exposition of the theory underlying Michell-type optimal layouts can be found in Strang and Kohn [90]. We note that the application of Michell-type structures in 3D is only appropriate for certain problem settings: e.g., Sigmund et al. [83] observed that, with no lower-bound constraints on shell thickness, variable thickness shells are likely to emerge as a solution. While their analysis is limited to compliance minimization, our experiments show it also applies to weight minimization (see Figures 3.21, 3.22, and 3.23).

Shape derivative methods. Shape-derivative based optimization techniques (e.g., Allaire and Jouve [5]) can obtain very good results when one needs to improve an existing design, by evolving the shape to a local minimum. However, while level-set methods of this type allow for topology changes, the result does vary considerably depending on the starting point. In contrast, our goal is to obtain a starting point that is close to the global optimum, as long as the desired structure has a relatively low volume.

Digital fabrication. The works closest to ours in this domain are Li and Chen [58], Tam et al. [92], and Tam [91]. These methods are based on constructing structures from stress lines on surfaces which, while different from the optimal fields we compute, are often a close approximation. The overall pipeline of the method of Li and Chen [58] is similar to ours: this work starts with a field, and construct trusses following the field by tracing lines from supports to loads. The method is limited to planar elasticity and demonstrated only for relatively simple structures.

Tam et al. [92] uses FDM to add material directly along the principal stress lines, on 2.5D surfaces. The main problems they solve are stress line generation and selection. They minimize strain energy subject to a maximal total print length (i.e max material) and a consistent maximum spacing between lines.

Applying topology optimization (SIMP and ground structure methods) to 3D printing applications is discussed in Zegard and Paulino [102].

3.3 Motivating examples

To motivate our method, we start with simple examples of qualitatively different behavior of optimized structures. With these examples, we demonstrate that in general, for in-plane loading, using a thicker surface is significantly more efficient that using narrow protruding ribs, and this solution cannot be obtained when using the convex volume approximations used in previous work.

The two key behaviors of shell-like optimal structures, observed in special-case analytic solutions and topology optimization (cf. [83]), are the formation of discrete narrow *ribs* protruding from the surface in bending-dominated cases (most forces are perpendicular to the surface), and relatively smooth variation in shell thickness in the pure tensile/compression case (in-plane forces), as shown in Figure 3.3.

These two behaviors are *not* observed in the simplified models of beam networks approximating a surface: in a typical network, beams do not expand in the direction parallel to the surface, to merge into a variable-thickness shell optimal in such cases. It turns out that this is due to the qualitatively inaccurate volume computation with the volume of the beam network approximated by the sum of individual beam volumes. We now consider two simple examples showing why this is the case.

First, we consider optimization of a single horizontal beam of width w and height h,

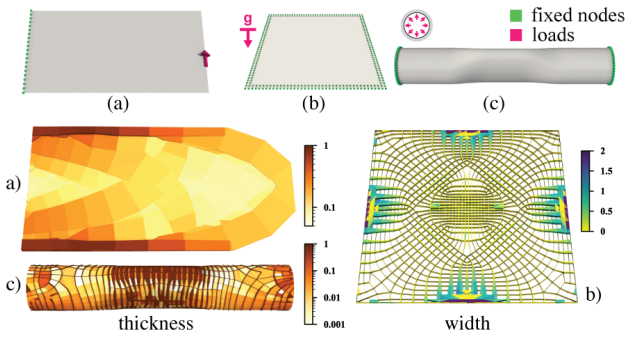


Figure 3.3: (a) optimal structure for a standard cantilever test case with variable thickness, cf. [83] (b) bending plate optimization, subject to uniform vertical loads, resulting in a rib structure qualitatively consistent with analytic results; (c) an optimized pipe structure, subject to internal pressure, exhibiting a mix of behaviors.

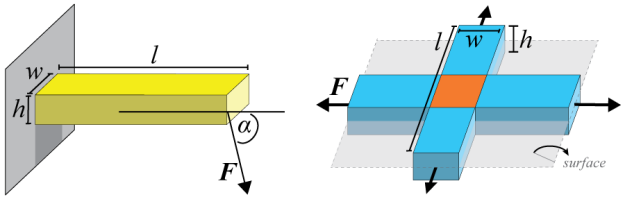


Figure 3.4: Left: beam loading and parameters; Right: surface approximated by two intersecting beams loaded in plane.

clamped at one end, and loaded at the other, at an angle α to the beam direction (Figure 3.4, left). This example clarifies optimal behavior when there is stress in only one direction, both for bending ($\alpha = \pi/2$) and tension ($\alpha = 0$). The second example involves two intersecting beams (Figure 3.4, right). It is a simple model for a piece of a surface where there is stress in two directions.

For a single beam, the force *along* the beam is proportional to the cross-section wh, and the force *perpendicular* to the beam (bending) is proportional to wh^3. The total force is proportional to $\cos\alpha w(h/l) + \sin\alpha w(h^3/l^3)$, where l is the beam length, and $\cos\alpha$ and $\sin\alpha$ are due to projection to the direction of F. For simplicity, assume the proportionality constant to be 1. We optimize the beam volume $V(w,h) = whl$, keeping the force balance constant: $\cos\alpha w(h/l) + \sin\alpha w(h^3/l^3) = F$. Eliminating w, yields an unconstrained minimization of $V = Fl^2/(\cos\alpha + y^2\sin\alpha)$, with $y = h/l$. When $\sin\alpha \neq 0$, the evident solution is to maximize

the relative thickness y: increasing thickness has a higher payoff as forces grow as h^3 vs. only as h for width. In contrast, when forces act along the beam ($\alpha = 0$), the constraint takes the form $wy = F$, i.e., the volume value is fixed and the choice of y makes no difference. We conclude that, for single beams, the solution can always be taken to be "thick and narrow" beams (we refer to them as *ribs*), with maximal possible thickness.

The situation is more complicated for surfaces. In the second example, we approximate the surface locally by beams aligned with the perpendicular stress directions (Figure 3.4, right). For simplicity, we assume the forces, widths w, lengths l, and thicknesses h to be the same for both beams. If we view the intersecting beams individually and approximate the total volume as $V = 2whl = 2wyl^2$, then the reasoning above applies to each beam: for in-plane forces ($\alpha = 0$) we get $wy = F$, and the optimal volume is $V = 2Fl^2$, independent of the choice of w. However, even a small bending component will prioritize maximal h solutions, so both beams will be thick and narrow.

Considering beams in separation ignores the fact that *the intersection area of the beams is counted twice*: this part of material is performing "double work", supporting loads in two directions along two beams. The correct combined volume of two beams is given by

$$V'(w, y) = 2wyl^2 - w^2yl,$$

assuming the same beam width and thickness for both. As before, for in-plane forces ($\alpha = 0$), we have the constraint $wy = F$. Replacing y, the functional V' can be expressed then as $V'(w) = Fl(2l - w)$. From this expression, it is clear the minimum volume is obtained maximizing *width*, as opposed to *thickness*. For a maximal width l, the optimal volume is Fl^2. In comparison, if we use a large relative thickness y, then optimal $w = F/y \approx \epsilon$ is close to zero, and the volume $V'(\epsilon)$ is close to $2Fl^2$, two times higher than optimal $V'(l)$.

More generally, a combination of bending and membrane forces is required to keep an

arbitrarily shapes structure stable (Figure 3.5). In this case, two intersecting beams with an out-of-plane load in addition to in-plane, there is an optimal trade-off between w and h minimizing the volume.

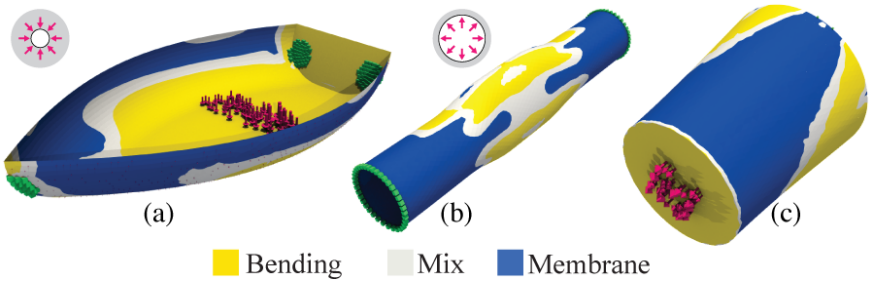


Figure 3.5: Loaded shells dominated by bending and membrane forces, and a mix of these. Support nodes are highlighted in green while loads are shown in red. Loads on (a) and (b) include, respectively, external and internal pressure.

If we further impose constraints on maximal and minimal surface thickness, even in tensile-dominated areas, ribs would form, because the optimal solid shell there would be too thin. A general optimization method should be able to smooth between grillage-like structures for bending dominated areas and "thin and wide" structures for the rest of the surface.

We conclude, from these examples, that to reinforce a shell in a manner close to optimal for arbitrary loads and shell shape, both solid variable-thickness and rib-like structures may be required in different areas of the surface, and for these to emerge, in a beam-based optimization problem, a non-convex volume function accounting for beam intersection areas has to be used.

3.4 Problem formulation

We start with a description of the variable-thickness perforated surface shell structure that we use to model surface reinforcement, and the optimization problem we aim to solve.

3.4.1 Parameters for reinforced shell structure

Our input is an initial shell M of thickness $h_s \geq 0$ (constant per triangle) represented by a triangle mesh, with a vector of external forces f applied to its vertices, and a set of fixed vertices (supports).

We aim to compute a reinforcement structure, added to the initial shell, which we call a *perforated shell of variable thickness* M^p (Figure 3.6). M^p consists of a partition $\mathcal{P}$ of the input surface into polygonal faces (typically quads), corresponding to 3D *cells*, and an extruded shape for each cell, consisting of *blocks* as described below. The blocks for sequences of cells may form rib-like structures, if the blocks are tall and narrow, or can fill the cells completely which corresponds to the variable-thickness solid shell case.

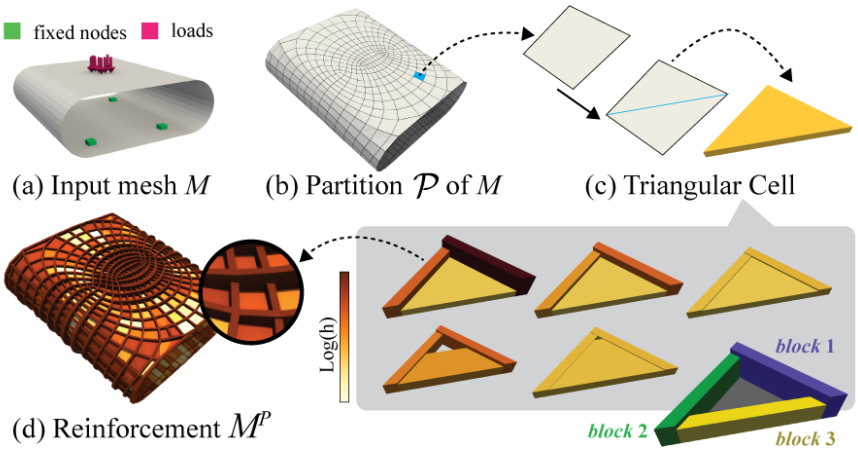


Figure 3.6: Perforated variable thickness shell. (a) Input triangle mesh M with boundary conditions; (b) Partition $\mathcal{P}$ of the input into polygonal faces; (c) each triangular subcells comprises three trapezoidal blocks; (d) extruded cells forming variable-thickness shell structure M^p;

Given an approximate user target for cell size, our goal is to optimize the edge orientation of the cell boundaries, and the thicknesses and widths of blocks forming each cell to minimize the weight, while maintaining an upper bound on stresses (calculated using a beam model). M^p can be viewed as the reinforcement structure for M.

To simplify our problem, we split all polygonal cells into triangular subcells. We refer to

the additional edges inserted in this way as *diagonals*. We treat these in a special way in the optimization, and in the end ensure that the triangular sub-cells can be merged back into the original polygonal cell (Section 3.7).

Cell geometry parametrization. For each edge of a face of M^p we introduce two parameters, width w_i and thickness h_i. With each edge, we associate a hexahedron *block* constructed by creating a strip inside the face at distance w_i from the edge and extruding the resulting trapezoid along the triangle normal by h_i (Figure 3.7). We model each block as a beam including tension/compression and bending forces. While this is a relatively coarse approximation of the shape, it allows obtaining an approximation of the solution robustly and quickly. These results can be further refined by shape optimization methods (e.g. Panetta et al. [71]).

Volume discretization. We consider triangular cells with sides l_i, $i = 1, 2, 3$, and blocks with rectangular cross-sections of width w_i and thickness h_i along the edge l_i (Figure 3.7). The simplest approximation of the volume which is typically used in low-volume truss models, when applied to our system would yield simply $\sum_i w_i h_i l_i$. However, as discussed in Section 3.3, this approximation of the volume results in highly sub-optimal results for shells regions dominated by tensile forces.

A more precise approximation is the volume of the extrusion of three trapezoidal regions by different heights. Denote the normalized width as $y_i = w_i/a_i$, where a_i is the height of the triangle with base on side l_i, and the triangle area as $A = \frac{1}{2} a_i l_i$ for any i. Given a permutation (i, j, k) of $(1, 2, 3)$ for which $h_i \geq h_j \geq h_k$, the volume is given by the following

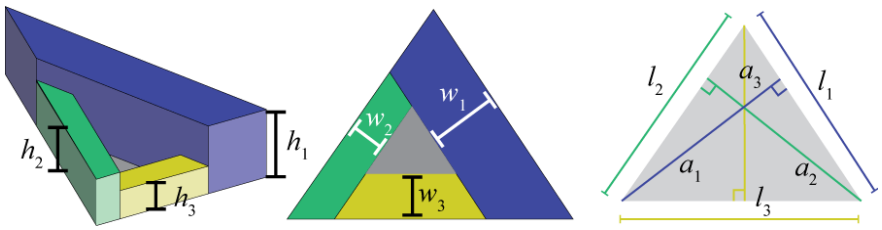


Figure 3.7: Geometric parametrization of triangular cell. Left: perspective view of the three blocks; Center: top view showing widths; Right: triangle sides l_i and respective heights h_i.

simple expression:

$$\begin{aligned} V(y,h) =& A\left((2-y_i)y_ih_i + (2-2y_i-y_j)y_jh_j \right. \\ & \left. +(2-2y_i-2y_j-y_k)y_kh_k\right). \end{aligned} \tag{3.1}$$

This volume can be written as $V(y,h) = \max_{(i,j,k)} V_{ijk}(y,h)$, where $V_{ijk}(y,h)$ is given by (3.1) for arbitrary permutation (i,j,k). This expression for the volume is useful for the optimization method in Section 3.7.

The out-of-plane heights h_i can be constrained not to exceed a user-defined value h_{max}, and the normalized widths y_i are constrained so that the trapezoidal areas do not overlap:

$$y_1 + y_2 + y_3 \leq 1, \quad y_i \geq 0, \quad 0 \leq h_i \leq h_{max}, \quad \text{for } i = 1,2,3. \tag{3.2}$$

3.4.2 Elastic deformation discretization

We model the perforated shell structure as a *beam network:* for each interior edge, there are two beams, corresponding to the blocks of incident cells along the edge.

Notation. The beam network consists of a set of beams $\mathcal{E}$ that are joined together at nodes (Figure 3.8). For a node i, $N(i)$ is the set of indices of nodes connected to it, and the vector $\mathbf{e}_{ij}$ connecting nodes i and j corresponds to the edge e_{ij}. For each edge, $\hat{\mathbf{e}}_{ij}$ denotes the unit

vector along $\mathbf{e}_{ij}$. The edge e_{ij} has length l_{ij}. We assume that all cells are made of uniform material with E as Young modulus. We use $\sigma[ij]$ and $\varepsilon[ij]$ to denote one-dimensional (tensile or bending) stresses and strains of beam connecting vertices i and j in a given cell. From now on, we use $(.)^t$ and $(.)^b$ to denote tensile and bending terms respectively.

Beam linear elasticity discretization. The tensile strain along each beam is the scalar $\varepsilon^t[ij] = (\mathbf{u}_j - \mathbf{u}_i) \cdot \hat{\mathbf{e}}_{ij}/l_{ij}$, same for both blocks at the edge e_{ij} where $\mathbf{u}_i$ is the displacement of a vertex i. It is related to the stress by $\varepsilon^t[ij] = \sigma^t[ij]/E$.

For our problem, bending discretization is critical. We use a pure displacement-based beam bending approximation, but a more standard beam element could be used. Our beams are *clamped* to a freely rotating plane at each vertex, i.e., preserve the angle between the beam and the (freely moving) normal to that plane. Each beam is also connected to the original input shell, resulting in lower torsion. We use a simple discrete beam model for bending, where we neglect the torsion, and define the bending strain e_{ij}, i.e., the change in the curvature of a beam, as

$$\varepsilon^b[ij] = \hat{\mathbf{e}}_{ij}^T(\Delta\hat{\mathbf{n}}_j - \Delta\hat{\mathbf{n}}_i)/l_{ij}, \tag{3.3}$$

where $\hat{\mathbf{n}}_i$ and $\hat{\mathbf{n}}_j$ are normals meeting at nodes i and j, and $\Delta\hat{\mathbf{n}}$ denotes linearized change of the normal. The normal change, in turn, is expressed in terms of the displacements $u_{i\ell}$, $\ell \in N(i)$ of the incident vertices.

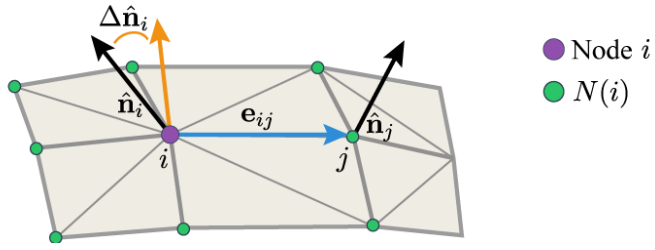


Figure 3.8: Beam bending discretization.

This leads to the expression for a scalar bending strain on beams:

$$\varepsilon[ij]^b = (D^b_{ij})^T u \tag{3.4}$$

with the expressions for D^b_{ij} given in Chapter Notes 3.10.4.

At a distance z from the middle surface of a beam, the strain is given by $\varepsilon^t + z\varepsilon^b$, where we omit the beam index. Based on the standard Bernoulli beam assumptions, after integration over $z \in [-h/2, h/2]$ the total beam energy can be expressed as

$$\frac{1}{2}(wlhu^T(D^t)(D^t)^T u + \frac{1}{6}wh^3 lu^T(D^b)(D^b)^T u),$$

which leads to the following expression for the stiffness matrix of the beam system:

$$K^B = wlh(D^t)(D^t)^T + \frac{1}{6}wh^3 l(D^b)(D^b)^T. \tag{3.5}$$

Given the expression for strain $\varepsilon^t + z\varepsilon^b$, clearly both strain and stress are maximal on one of the surfaces, i.e., for $z = h$ for a given beam. This leads to the following stress constraint:

$$|(D^t_{ij} + h_{ij} D^b_{ij})^T u| < \sigma_0. \tag{3.6}$$

The stiffness of the *reinforced* system combines the stiffness of the beams and shell. We form the shell stiffness matrix K^S using the elastic discretization described on Section 3.6. The combined stiffness matrix K is given by $K = K^B + K^S$.

Optimization problem. We use index c for triangular cells. Let w be the vector of width parameters of all cells, y the vector of corresponding normalized widths w_i/a_i, h the vector of all thickness parameters, H the diagonal matrix with thicknesses on the diagonal, u the

displacements, f the forces, and $\mathbf{1}$ be the vector of all ones.

We now formulate our optimization problem:

$$\begin{aligned}
&\min_{h,y,\mathcal{P}} \sum_{c\in cells} V(h_c, y_c) \quad \text{s.t.} \quad K(h, y, \mathcal{P}) = \{, \\
&|(D^t \pm D^b H)^T u| \le \sigma_0 \mathbf{1}, \quad 0 \le h \le h_{max}, \quad y \ge 0, \\
&y_1^c + y_2^c + y_3^c \le 1, \text{ for all cells } c,\ i = 1, 2, 3,
\end{aligned} \tag{3.7}$$

where the absolute value in the stress constraint is taken elementwise, and the minimum is taken over all partitions $\mathcal{P}$ of M. The cell volume V is defined in (3.1), the stress constraint comes from (3.6), and the last three constraints correspond to (3.2). Additionally, we enforce the same thickness on two sides of all cell diagonals.

As optimization over all possible partitions into cells is an intractable problem, combining combinatorial and continuous aspects, we use an heuristics to decide the partition first using *beam continuum approximation* (Section 3.6). Once $\mathcal{P}$ is fixed, we optimize with respect to w and h only.

While the above formula volume differs only by a seemingly simple quadratic term from the simplest approximation, this completely changes the behavior of the problems, and, in particular, the behavior of the solvers. The problem no longer reduces to convex by a change of variables as it is the case for the simplest formula (cf., e.g., Hemp [41]), and a different type of solvers need to be applied. In our experiments, commonly used general purpose non-convex solvers converge very slowly and often fail to make progress. Our solution is described in Section 3.7.

3.5 Overview of the approach

Our pipeline for solving the optimization problem consists of the steps listed below.

1. **Field optimization.** Compute a per-triangle cross-field on surface using stress-based optimization (Section 3.6). This field corresponds to an idealized system of densely spaced thin beams (*beam continuum*) with directions chosen to minimize weight. The problem is formulated in terms of displacements and the desired cross-field is obtained from the symmetric strain tensors. This requires solving a convex optimization problem with inequality constraints.

2. **Quadrangulation.** Create a quad-dominant mesh aligned to this cross-field with a user-controlled spacing of edges. This step is done using a version of mixed-integer quadrangulation [18], although any quad layout method with field alignment can be used instead (Section 3.7). The faces of the mesh will correspond to the *cells* of the perforated shell M^p.

3. **Cell optimization.** Optimize shape parameters of the cells (Section 3.7). We introduce a substructure for each cell, with a small number of control parameters defining its shape (widths w and thicknesses h of rectangular beams along each side). We derive the optimal material distribution by efficiently solving a nonlinear, non-convex problem minimizing the total volume of all cells with respect to w and h, while keeping stresses below a user-defined maximum. To make the problem tractable we defined an efficient local-global optimization method.

4. **Final geometry construction.** Finally, we derive the final geometry of M^p according to optimized widths and thickness. We obtain a triangle mesh by performing a sequence of boolean operations between meshes representing the beams. The final watertight and manifold mesh can be directly used for 3D printing or decomposed into elements for FEM analysis.

In the following sections, we describe the details of the steps, in the order in which they are

applied to produce the final result; however, the key step is the third one (*cell optimization*), as it has most impact on the optimality of the result, as demonstrated in the evaluation (Section 3.8).

3.6 Weight-minimizing field optimization

In this section, we describe our method for constructing a field of directions on the surface which approximates optimal directions for weight minimization with bounded stress. The cells in our construction will be aligned with these directions.

The key idea is to solve a version of the beam weight minimization in the limit case. We assume that there is a *continuum* of infinitely thin and infinitely close beams in two orthogonal directions forming the surface. The density of the beams and their orientations are the optimization variables. The idea of using this type of continua goes back to Michell [63]; in contrast to the standard Michell continua formulations, which takes only tension into account, we use both tension and bending. We describe first the classical theory of Michell continua, leading to a convex problem, and then extend it to the case of shells with bending forces, preserving convexity.

3.6.1 Michell continua

Here, we briefly review the classical solution, following Strang and Kohn [90]. The best directions are known to be the principal stress directions of the *optimized* structure. (These are *not* the same as the stress direction on the original input shell, although these fields are often close.)

The force balance for a plate or a shell with no bending is given by the standard equations in terms of in-plane stress tensor σ, strain ε, possibly varying elasticity tensor $E(\mathbf{p})$, and

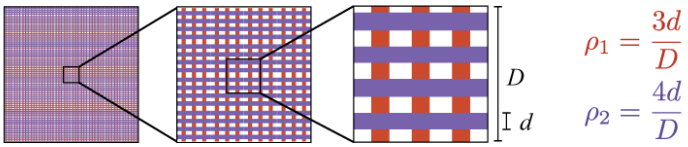


Figure 3.9: Michell continua example. In the limit of small thicknesses d, every point is characterized by two orthogonal beam densities, ρ_1 and ρ_2.

external force density $\mathbf{f}$:

$$\text{div}\sigma = \mathbf{f},\ \sigma = E(\mathbf{p}) : \varepsilon;\ \varepsilon = \frac{1}{2}(\nabla \mathbf{u} + \nabla^T \mathbf{u}), \tag{3.8}$$

where $A : B = \sum A_{ijkl} B_{kl}$ for a 4-tensor and a 2-tensor.

A Michell continuum is an idealization of a beam network, characterized, at every point, by beam densities ρ_1 and ρ_2 in two directions (Figure 3.9). In other words, how many beams cross a unit-length segment along one of the coordinate directions. In the limit of small thicknesses, the total fraction of a small area covered by trusses at a point $\mathbf{p}$ is $\rho_1(\mathbf{p}) + \rho_2(\mathbf{p})$. The total volume of the trusses in the continuum can be defined as $\int_\Omega \rho_1 + \rho_2 dA$. Note that this approximation of area covered by trusses suffers from the same flaw pointed out in Section 3.3.

The optimal trusses have to be oriented along stress directions, and be critically stressed, i.e., all (non-averaged) stresses on the trusses are equal to maximal stress σ_0. This leads to the relationship between ρ_i and corresponding averaged principal stress: $\lambda_i(\sigma) = \rho_i \sigma_0$, $i = 1, 2$, where $\lambda_i(\cdot)$ denotes the i-th singular value.

Then, we obtain the following optimization problem for the volume, formulated entirely in terms of stresses:

$$\text{minimize} \int_\Omega |\lambda_1(\sigma)| + |\lambda_2(\sigma)| dA,\ \text{subject to div}\sigma = \mathbf{f}. \tag{3.9}$$

This problem is known to be *convex* [90] (although it is more difficult than the linear

programming formulation for a truss network). Note that principal stress directions are not fixed and are determined by the optimization. We use these directions as the field for orienting beams in M_p.

The problem (3.9) has a simple dual (Chapter Notes 3.10.2) of the form

$$\text{maximize} \int_\Omega \mathbf{f}^T \mathbf{u} dA \text{ subject to } |\lambda_i(\varepsilon)| \leq \varepsilon_0, i = 1, 2, \tag{3.10}$$

where ε is the strain of the optimal solution (Figure 3.10). The dual problem is significantly easier to deal with in the case of continua.

We note that here we neglect the overlap volume of trusses discussed in Section 3.3; while it could be included as $-\rho_1\rho_2$ term, this would immediately make the problem non-convex, and the benefit of more precise field optimization in terms of volume reduction is minor (Section 3.8).

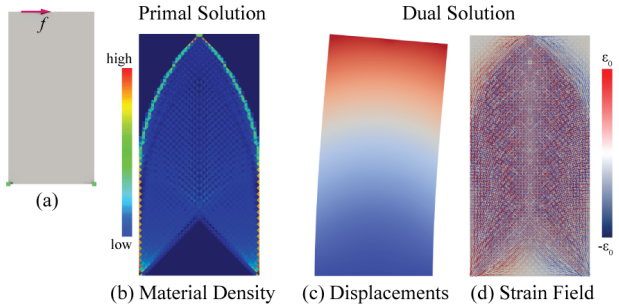


Figure 3.10: Field optimization: (a) Design domain Ω and boundary conditions; (b) Primal solution, squared densities $\sqrt{\rho_1 + \rho_2}$; (c,d) Dual solutions, displacements u and strain ε eigenvalues and vectors.

3.6.2 Continuum optimization with bending

Next we generalize problem (3.10) to include bending forces.

If the thickness of the shell remains fixed, one can add bending to the functional with relative ease without changing convexity of the problem. We set the shell thickness in this

case to half of the maximal allowable thickness; while the resulting field is suboptimal, as we show experimentally in Section 3.8, inaccuracy in the beam direction has less effect on the overall weight reduction, compared to width/thickness optimization of beams.

We make the standard assumption of planar stress for the shell, i.e., no stresses are active in the direction perpendicular to the shell surface. The strain at a distance z from the midline of the shell is given by (Figure 3.11)

$$\varepsilon(z) = \varepsilon^t + z\varepsilon^b,$$

where ε^b is the *bending strain* tensor , equal to the linearization of the change in the shape operator $\nabla\hat{\mathbf{n}}$.

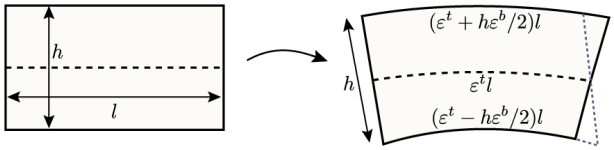


Figure 3.11: Vertical strain distribution in a shell.

Consistently with the Michell continuum, we seek to minimize the total weight of a beam continuum, bounding the stress everywhere by σ_0. We observe that the eigenvalues of a 2×2 symmetric matrix A, using the substitutions $a = (A_{11} + A_{22})/2$, $b = (A_{11} - A_{22})/2$, and $c = A_{12}$, are of the form $a \pm \sqrt{b^2 + c^2}$. Note that these are respectively convex and concave functions of the argument, and, therefore, reach their maxima (respectively minima) on the boundary of the shell, for $z = \pm h/2$. For this reason, it is sufficient to bound eigenvalues of stress (or strain) only for $z = h/2$ and $z = -h/2$, to guarantee the bounds elsewhere.

In the case of bending, the dual problem formulated in terms of displacements has a simpler form relative to the primal problem:

$$\text{maximize } \int_\Omega \mathbf{f}^T \mathbf{u} dA \text{ subject to } |\lambda_i(\varepsilon^t \pm \frac{h}{2}\varepsilon^b)| \leq \varepsilon_0, i = 1, 2. \tag{3.11}$$

Note that now there are two sets of constraints, corresponding to two surfaces of the shell.

The strain tensors $\varepsilon(h/2)$ and $\varepsilon(-h/2)$ can be interpreted as defining cross-fields on the surface. We use the angular average of these fields to align the cells.

3.6.3 Field discretization and optimization

Finally, we describe a discretization of the convex problem defined above, and how to use the resulting field to build a mesh.

As a first step, we solve a discrete version of the problem (3.11), which yields displacements $\mathbf{u}$ at vertices. From these displacements, we compute the per-triangle strain field eigenvectors, forming a cross-field on the surface, i.e., an assignment of 4 unit vectors, aligned with perpendicular principal strain directions, to each triangle.

Discretization of the optimization problem. The optimization problem (3.11) has a relatively simple discretization that can be readily plugged in into a cone program solver. e.g., MOSEK [7].

We assume that the surface is given as triangle mesh, $M = (V, E, F)$, and the same notation is used for edge vectors and vertices as we used for beam networks.

The variables in the problem are displacements, which we discretize using standard piecewise-linear functions on the surface, with the vector of unknowns u (we use non-bold letters for high-dimensional vectors including all components of corresponding three-dimensional quantities).

The two quantities that need to be discretized are tensile and bending strains; we define these per triangle.

If $\mathbf{e}_{ij}$ are the vectors along triangle edges, for a triangle T, we have the following expression

for the strain, computed as $\frac{1}{2}(\nabla u + \nabla^T u)$:

$$\varepsilon_T^t = \frac{1}{4A_T}\left(\sum_{i=1,2,3} \mathbf{e}_{jk}^{\perp}\mathbf{u}_i^T + \mathbf{u}\mathbf{e}_{jk}^T\right), \tag{3.12}$$

where A_T is the triangle area, j is the vertex after i in CCW order, k is before i, and $\mathbf{e}_{jk}^{\perp}$ is the vector in the triangle plane perpendicular to the side.

If, by abuse of notation, we use ε_T^t to denote the vector of three distinct components of the strain tensor, we can write the expression in the form $B_T^t u$, where u is the vector of all displacement degrees of freedom.

To discretize the bending strain, we use the triangle-based approximation of the shape operator, following the overall idea in Oñate et al. [69] and Grinspun et al. [35], using vertices of the edge-adjacent triangles to compute the changes of the average normals at edge midpoints, and computing the gradient of the normal using the formulas (3.12).

This leads to the following expressions for the bending strain on triangles, in which we neglect the triangle deformation: in the deformation modes for which the bending strain is high (i.e., if the curvature changes a lot, in-plane deformations are small). We use the following formula from Grinspun et al. [35], Figure 3.12:

$$\varepsilon_T^b = \sum_{i=1,2,3} \frac{\theta_i}{2Al_i}\mathbf{e}_i^{\perp}(\mathbf{e}_i^{\perp})^T \tag{3.13}$$

where θ_i are linearized *changes* in the angles between normals of adjacent triangles.

Similarly to ε^t, we can write $\varepsilon^b = B^b u$. Then the discrete problem takes the form

$$\text{maximize } f^T u \text{ subject to, for all T, } |\lambda_i(B_T^t \pm hB_T^b)u)| \le \varepsilon_0, i = 1, 2. \tag{3.14}$$

where f, similarly to the beam case, denotes the vector of per-vertex forces, and u is the

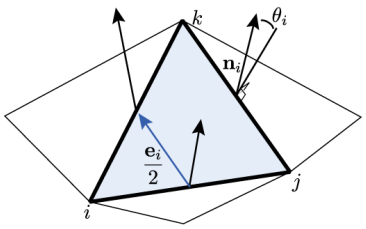


Figure 3.12: Discretization of the bending field.

vector of all vertex displacements. We use eigenvalue formulas defined in Chapter Notes 3.10.1 for (3.19) to convert the problem to a convex cone problem, which we solve using the MOSEK solver [7].

Detecting field zones. While the output of the previous step defines a tensor for each triangle, not all of these are meaningful. In some cases (if the triangle is not deformed at all, or deformed negligibly) the strain is zero. More generally, some points may have isotropic strains of the form kI, where k is a nonzero constant, for which all vectors are eigenvectors, so the cross-field is not defined uniquely on this triangle. For general fields, such points are usually isolated. However, for the fields corresponding to the solution of the problem we are considering, the situation is different. There are four possible regimes (see, e.g., Strang and Kohn [90]). Specifically, the possibilities include

1. $\lambda_i(\varepsilon) = -\lambda_j(\varepsilon) = \varepsilon_0$, principal strains are critical and have opposite directions; this corresponds to well-defined two orthogonal beam families;

2. $\lambda_i(\varepsilon) = \lambda_j(\varepsilon)| = \pm\varepsilon_0$, principal strain are critical and have the same direction; in this case, stresses (which are dual variables to the inequality constraints) are large but beam *directions* are not well defined;

3. $|\lambda_i(\varepsilon)| < |\lambda_j(\varepsilon)| = \varepsilon_0$, only one strain is critical; this corresponds to a single family of beams.

4. $|\lambda_{i,j}(\varepsilon)| < \varepsilon_0$, in this case, stresses are both zero, which means there are no beams in this area.

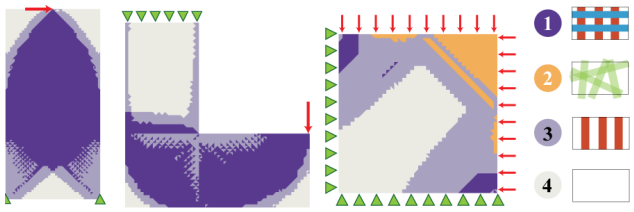


Figure 3.13: Zones of an optimal strain field. (1) Two orthogonal directions, (2) no preferred direction, (3) one direction, (4) no beams. The crossfield directions are well-defined only for regions 1, 3.

In cases 1 and 3, the crossfield directions are well-defined (purple zones on Figure 3.13). In cases 2 and 4 these are either not defined or are not relevant, due to the absence of structure. For this reason, for our construction, we use only regions 1, and 3, which we detect by requiring at least one eigenvalue to be close to ε_0, and the difference of eigenvalues to be more than a constant.

We call the resulting field *salient*. The situation is essentially identical to the cross-fields used for constructing quadrangulations: typically, a field aligned with curvature directions is used as a starting point, and only in areas where the difference of two principal curvatures is high.

Completing the field. To complete the field on the whole surface, which is needed for a complete structure, we use cross-field constrained optimization procedure of Bommes et al. [18]. In this algorithm, the cross-field is encoded by a per-triangle angle, with respect to a reference direction β_i in each triangle. The angles on salient triangles are fixed. On the remaining triangles, these are found by a greedy solve of a mixed-integer problem minimizing the energy

$$E = \sum_{\text{edges}\,(ij)} (\beta_i - \beta_j + k_{ij}\frac{\pi}{2} + \kappa_{ij})^2$$

where the summation is over all dual edges connecting triangles i and j, κ_{ij} is the angle between reference directions in triangles, and k_{ij} is an integer unknown accounting for the fact that cross-field values represented by angles $\beta + k\pi/2$ are the same.

In the resulting field, defined by the angles β_i for all triangles, and integers k_{ij} for all edges, one can easily compute per-vertex field index and identify field singularities, which become irregular (valence different from 4) vertices of the quad mesh at the next step. We refer to Bommes et al. [18] for details of the index computation.

3.6.4 Construction of the quad-dominant network

In general, there may be no optimal beam spacing (in the low-volume limit, the finer the structure, the lower the optimal volume can be for a given stress). For this reason, the beam spacing is defined by a user-specified parameter H. The most direct approach for constructing a quad mesh aligned with a field would be to trace it. However, while it was shown [68, 76] that this approach can be implemented robustly, in general it requires T-joints (i.e. beams joining other beams in the middle), and it is in general hard to ensure uniform spacing over the whole mesh. We choose a more conservative approach based on constructing a conforming quadrangulation, without T-joints, using a version of the mixed-integer quadrangulation algorithm [18] at this step.

While the method does not guarantee perfect alignment of the parametric lines to the input field, it minimizes the deviation in least-squares sense. We refer to Bommes et al. [18] for further details.

3.7 Optimizing cell geometry

Given an input quad-dominant mesh, split into triangles, we aim at deriving the optimal width and thickness for each edge. As we previously stated, to obtain a structure close to

optimal, the edges need to follow the directions we derived in the field optimization step. While our material distribution optimization method works for any mesh, the further the edges deviate from their optimum directions the greater would be the total weight. The main idea of our algorithm is to use a *local-global* iteration, solving per-triangle *concave* problems for each cell, for which the solution is guaranteed to be on the boundary on the constraint domain. This yields a rapidly converging efficient algorithm.

3.7.1 Optimization algorithm

We introduce a domain-decomposition-style algorithm for solving the problem (3.7). We observe that in the optimization problem (3.7), all constraints except $Ku = f$ are localized, i.e., each constraint uses variables related to one cell. Moreover, the functional itself is a sum of local volume terms $V^c(w^c, h^c)$. $Ku = f$ expresses the equation of force balance, i.e., that the sum of beam forces at each node is equal to the external force at this node. Our approach is to fix individual beam forces to their values for current values of geometry parameters w and h, and then solve for an update to w and h as a set of local volume-minimization problems with variables w^c, h^c, replacing the global constraint $Ku = f$ with local constraints requiring that individual beam forces remain the same.

We start with an outline and then elaborate on how the local step optimization problems are solved.

Initially, we assign sufficiently high values to w and h, to ensure that max stress constraints are satisfied.

- **Global step.** The global step is just the standard solve of the elastic equilibrium problem, for fixed cell parameters: *Solve $Ku = f$, for fixed K defined by w and h. Compute beam forces as described below.*

- **Local step.** The local step is the key part of our algorithm. Recall that an important

feature of optimal structures is that they are *critically stressed* i.e., the maximal stress on any element is equal to the maximum possible σ_0. The reasons for this are straightforward: if a stress on an element is below zero, one is free to remove some material, increasing the stress on the remaining part, and decreasing the weight.

This motivates our approach. For the local step, we keep the displacements computed at the global step fixed and solve for widths and thicknesses, that would result in maximal stresses on blocks reaching the critical value σ_0 for given displacements. Each system has 6 unknowns, with 3 constraints on stresses.

Block forces. To formulate our local algorithm, we introduce *block forces*, for individual blocks of each cell; we determine w^c and h^c for each cell by minimizing the cell volume, while keeping the block *critically stressed* i.e., with maximal stress σ_0 and block forces constant.

Let K^{loc} be the element stiffness matrix corresponding to a block B. The vector of forces corresponding to a beam is the vector $\nabla_u E^{loc}$, i.e. the derivative of the block energy $E^{loc} = \frac{1}{2} u^T K^{loc} u$ with respect to displacements. Most forces in this vector are zero.

We use lower-case, non-bold d^t for the column vector of $(D^t)^T$, and d^b for column vector of $(D^b)^T$, corresponding to the stresses on block B. After some rearrangement, the force vector $f^{loc} = K^{loc} u$ due to elastic forces produced by a block is

$$f^{loc} = K^{loc} u = Ewhl \left((d^t)^T u d^t + \frac{1}{6} h^2 (d^b)^T u d^b \right).$$

Note that this equation implies that f is in the span of vectors d^t and d^b.

Let $\tilde{d^t}$, $\tilde{d^b}$ be the dual pair of vectors to d^t, d^b. Let the magnitudes of tensile and bending stresses be $|E(d^t)^T u| = \sigma^t$, $|E(d^b)^T u| = \sigma^b$; by taking dot products of both sides with the

dual vectors $\tilde{d}^t$, $\tilde{d}^b$, we arrive at the equation (dropping beam/cell subsripts):

$$wh\sigma^t = |f^t/l| = g^t,\ \frac{1}{6}wh^3\sigma^b = |f^b/l| = g^b. \tag{3.15}$$

Local optimization problem. The stress in the block, under our assumptions, reaches its maximal value at the top or bottom, where its magnitude is equal to $\sigma^t + h\sigma^b$. This leads to the critical stress constraint $\sigma^t + h\sigma^b = \sigma_0$. Using expressions for g^t and g^b above, which we keep fixed at the local step, this is equivalent to

$$\frac{g^t}{h} + \frac{6g^b}{h^3} = \sigma_0 ay,$$

where we have switched to the variable $y = w/a$ introduced in (3.7), where a is the corresponding cell triangle height. Without loss of generality, we assume σ_0 to be 1, which can be achieved by scaling all forces. The complete local problem in variables h_i^c, w_i^c, $i = 1, 2, 3$ is:

$$\begin{aligned}
&\min_{y^c, h^c} V(y^c, h^c) \quad \text{s.t.} \quad \frac{g_i^t}{h_i^c} + \frac{6g_i^b}{(h_i^c)^3} = \sigma_0 y_i^c a_i \\
&0 \le h_i^c \le h_{max}, \quad y_i^c \ge 0, \quad \text{for } i = 1, 2, 3 \\
&y_1^c + y_2^c + y_3^c \le 1,
\end{aligned} \tag{3.16}$$

where the volume V is given by (3.1), and the last three constraints by (3.2).

By eliminating the stresses, we arrive at a *single constraint per block* relating w and h, which we express as follows:

$$y = (6g^b z^2 + g^t z)/a, \tag{3.17}$$

where $z = h^{-1}$ is a new variable we introduce to simplify the expressions. This allows us to eliminate all variables w_i^c from the local optimization problem, leaving only three variables z_i^c, with constraints $0 \le z_{min} \le z_i^c$, $i = 1, 2, 3$, and $z_{min} = (h_{max})^{-1}$.

We say that a cell is *filled* if the equality $y_1^c + y_2^c + y_3^c = 1$ is satisfied, i.e. the blocks completely fill the cell.

Without loss of generality, we assume that for the solution $z_1 \leq z_2 \leq z_3$; in practice, 6 problems corresponding to 6 permutations of $(1, 2, 3)$ need to be solved and minimal solution picked.

Proposition 1. *The function $V(z^c)$ is a concave function of z_i. As a consequence, its minima are reached on the boundary of the constraint domain; specifically, it is reached at one of the five types of configurations:*

1. *all three blocks have maximal thickness: $z_i^c = z_{min}$, $i = 1, 2, 3$;*
2. *the cell is filled, i.e. $y_1^c + y_2^c + y_3^c = 1$, and no inequality constraint reaches equality;*
3. *the cell is filled, and two thicker beams have equal thickness, $z_1^c = z_2^c$;*
4. *the cell is filled, and two thinner beams have equal thickness, $z_2^c = z_3^c$;*
5. *the cell is filled, and the thickest beam has maximal thickness $z_1^c = z_{min}$.*

In the first case, the solution is completely determined. In the second case, there are four possibilities: no inequality constraint is active (a 2-variable unconstrained optimization problem needs to be solved, e.g., parametrized by z_1^c, z_2^c); the other three cases define one-parametric families of solutions, and one-dimensional unconstrained optimization needs to be performed to find exact values, as we explain below. These families can be parametrized by, e.g., $z_3^c = (h_3^c)^{-1}$, with the values of the remaining z_i^c and y_i^c determined from the active constraints. The proposition is proved in Chapter Notes 3.10.3.

This behavior of V is in stark contrast to the low-volume formulation ignoring common areas of beam-like parts of the structure: one can see that in three cases out of four, it creates a completely filled cell.

Solving the optimization problem. Proposition 1 leads to an efficient algorithm for the local step.

Observe that the constraint $y_1^c + y_2^c + y_3^c = 1$ has the form

$$\sum_i g_i^t z_i^c + 6 g_i^b (z_i^c)^2 = 1, \tag{3.18}$$

i.e., it is quadratic in z_i. This allows us to reduce the problem to a set of unconstrained optimization problems in one or two variables.

1. Compute g_i^b, g_i^t, $i = 1, 2, 3$, from (3.15) for current displacements.

2. Evaluate $V(z)$, for the case 1 solution with $z_i^c = z_{min}$.

3. For each permutation of $(1, 2, 3)$ solve three one-dimensional optimization problems, minimizing $V(z^c)$, for each of the cases 3-5, and the two-dimensional problem for case 2 of proposition Prop 1. In each case 3-5, substitute the active constraint for z_i^c into (3.18), yielding a quadratic equation in two remaining free variables, one of which is z_3^c. Solve it to express the other variable in terms of z_3^c, and solve a one dimensional optimization problem for $V(z_3^c)$. This yields a set of *solution candidates*; the minimal solution is guaranteed to belong to it.

4. Pick the minimal solution from the set of solutions obtained for all possible permutations and cases on the previous step.

5. Update w_i^c using formula $w_i^c = (6 g_i^b z_i^c + g_i^t z_i^c)$, and recompute the global stiffness matrix K.

The convergence behavior of the method is considered in Section 3.8.

Handling polygonal cells and postprocessing. There are three factors not considered in the solution method above: (1) possible inconsistency of thickness values across diagonal edges inside triangulated polygonal cells; (2) the coherence of block widths and thicknesses along the edge lines of the quad mesh, approximating the optimized stress lines (Figure 3.14). While jumps in thickness/width along these lines do not affect the stresses in our simplified model, in practice, these are likely to lead to localized stress concentrations close to jumps, and they are aesthetically objectionable. (3) stress values may slightly exceed the maximal stress after a final global step.

We experimentally observe that many of the candidate solutions have close values, especially in areas with no predominant stress direction.

We address (1) primarily in the process of optimization, at step 4, we pick a minimal candidate solution with lowest block thickness on the diagonal, which may not be the most optimal one, as long as it does not deviate above a threshold. Once the optimization is complete, for each subcell of a polygonal cell we increase the lowest block thickness to the maximal minimal thickness over the whole cell. In addition, for each cell, we store a number of candidate solutions with the smallest volumes.

We address (2) in a post-processing step using stored candidate solutions: for each non-diagonal edge of a cell, we find its continuation edges along quad mesh edge line in both directions, and choose the candidate solution closest in width to the average of the previous and next edge widths along the edge line.

To address (3), we find all blocks with stress exceeding $\sigma_0 = 1$, and increase their thickness and width, while maintaining constraints, to decrease the stress to the bound. This process is repeated iteratively until convergence. We note that all additional steps are designed to ensure that the final result satisfies stress constraints: in all cases, we never decrease the amount of material in cells, so while the resulting solution may be suboptimal, it always satisfies stress constraints. In practice, the effect of these alterations on the resulting weight

is small.

Alternatively, to alleviate these issues, one could optimize for the width and thickness per-vertex instead of per-edge. To be more precise, we would require two width/thickness per-vertex since two beams crossing at a vertex may be different. While this approach is, in principle, possible, in practice is much more expensive: for vertex-based cells, the volume function is not likely to be concave, so a general nonconvex solver needs to be used. Our experiments showed that, as the number of variables and constraints increases, a general solver is likely to get stuck in local minima (see Section 3.8 Figure 3.19).

Inflation. Once we have performed the weight optimization, we have one value of thickness and one value of width for each half edge of the optimized polygonal mesh. If we simply extrude the solid block that matches thickness and width for each half edge cell we end up in the situation illustrated in Figure 3.14.a, where there are visible discontinuities between adjacent blocks, causing possible structural discontinuities. Instead, for each continuous stream of quad edges, we generate a unique solid block that interpolates thickness and width along its length (Fig. 3.14.b). In detail, given a sequence of aligned edges, we first derive a thickness value for each vertex by averaging the thickness from its adjacent half edges. Similarly, we interpolate widths, but this time we derive two different values for each vertex, one for each side of the sequence. We then define a tangent vector for each vertex as the cross product between its normal and its direction along the edge sequence (obtained by averaging the direction of the two incident edges). Then, having defined a proper reference frame, a thickness, and a width for each vertex, we have all the information to extrude a proper volumetric block. We perform a boolean operation to merge all the blocks together to a manifold watertight mesh using the approach of Zhou et al. [105].

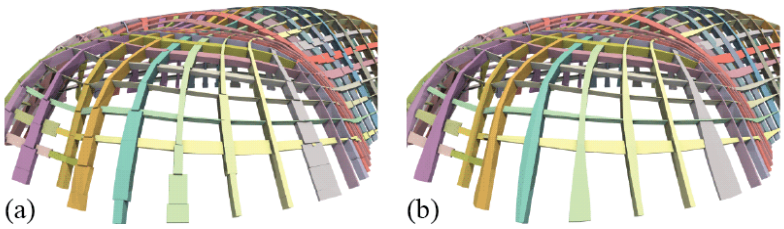


Figure 3.14: Inflation: (a) Each block is meshed independently, leading to visible artifact; (b) An entire stream of edges is meshed to produce a smoother result

3.8 Evaluation

Topology optimization. To validate our approach vs. a general-purpose topology optimization method, we solve a similar problem with topology optimization code [1] by restricting the volume of the material to a cylinder of fixed small thickness, and choosing the volume grid resolution to be half of the cylinder thickness, leaving little room for shell shape variation. We observe that for small target volume fractions, as expected, structures emerging in topology optimization are similar to the beam structures we construct (Figure 3.15), with similar volume fractions. For a large volume fraction, the topology optimization method results in variable-thickness shells, but in this case, due to severe constraints on the thickness of the shell, this does not make a significant difference. Default parameters were used in Aage et al. [1]. We note that topology optimization requires high resolution (with multiple cells fitting in the thickness direction of the shell) to achieve this type of results.

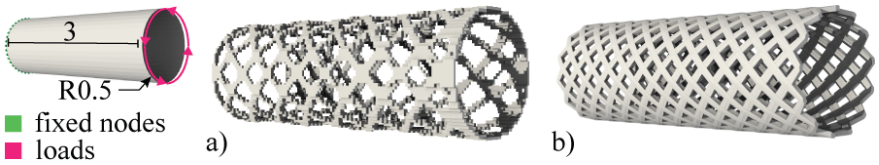


Figure 3.15: a) cylinder beam structure obtained using topology optimization with a target volume of 0.375; b) structure obtained using our method with a maximum thickness of 0.05, minimum thickness of 0.04, no support shell ($h_s = 0.00$).

Figure 3.16 shows the results of SIMP topology optimization vs. our method, with compliance as a function of the volume fraction. We observe similar behaviors for both

methods. We note that in our case we need to choose the beam spacing parameter: when this is chosen too coarse, the performance will deteriorate.

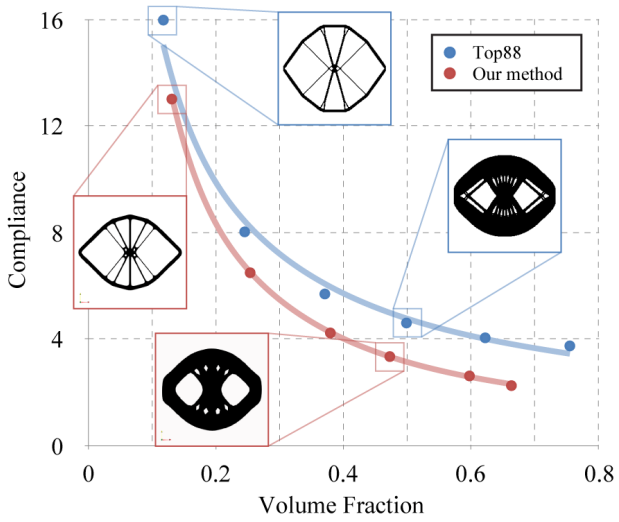


Figure 3.16: Volume fraction vs. compliance energy for a standard example, a cantilever beam, SIMP topology optimization vs. our method.

The role of beam direction. In Figure 3.17, we explore the dependence of the role of beam direction in structure optimality, by comparing structure volume for several fields, in addition to our optimized field (Eq. 3.14). As a "worst-case" baseline we use the cross-field at a maximal distance from the original field (d); as one can see, the field makes a significant difference when it is very far away from the optimal directions, i.e., the choice of directions matters. Similarly, the curvature field (c), unrelated to stress directions, produces relatively high values of the volume. On the other hand, the difference between the optimized field and the stress field (a and b), while present, is not large in most cases. This experimentally justifies using, e.g., Li et al. [57], the elasticity stress field instead of the optimized field; however the additional expense of using an optimized field is minimal, so there is effectively no penalty for this improvement.

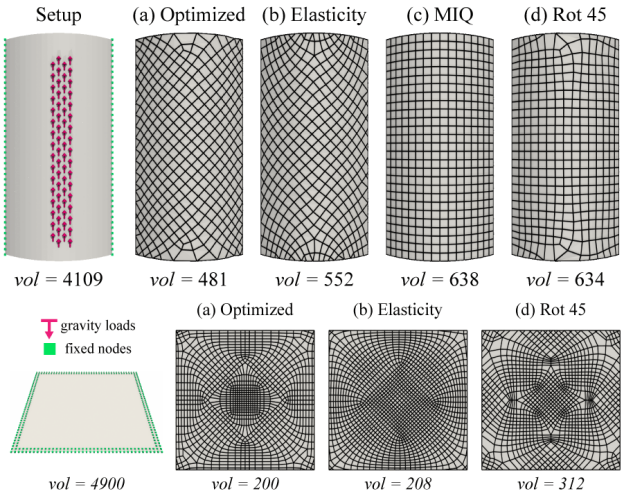


Figure 3.17: Comparison of volumes obtained using different fields for the shell structure. Images (a-d) show the quad-mesh obtained with the different fields: (a) optimized stress directions from solving (3.14), (b) stress directions from solving elasticity, (c) MIQ field constructed using the smoothing method of Bommes et al. [18], and (d) constructed rotating (a) by 45 degrees.

Convergence and dependence on the starting point. We have observed that our algorithm almost invariably converges in several iterations, and yields the expected behavior of solutions in the extreme cases (ribs in the case of bending-dominant shells, and wide-and-thin beams in high-tension/low-bending areas. The plots on Figure 3.18 show the volumes at each iteration for a basic and more complex problem. Note that sometimes the optimal value is approached from below: the local step overshoots the volume reduction and the stress exceeds the maximal allowed level. Nevertheless, the method recovers reliably.

Comparison to other optimization methods. We also compare our method to two general-purpose constrained optimization methods, SLSQP [53] and Ipopt, a barrier interior point method [21]. In this setup, we used an approximation of the beams volume that is smoother (it uses average instead of max) and simple lower and upper bounds on the width and thickness. As a starting point, we have used the solution of the convex problem with volume ignoring the overlaps. Somewhat surprisingly, these methods were not able to change

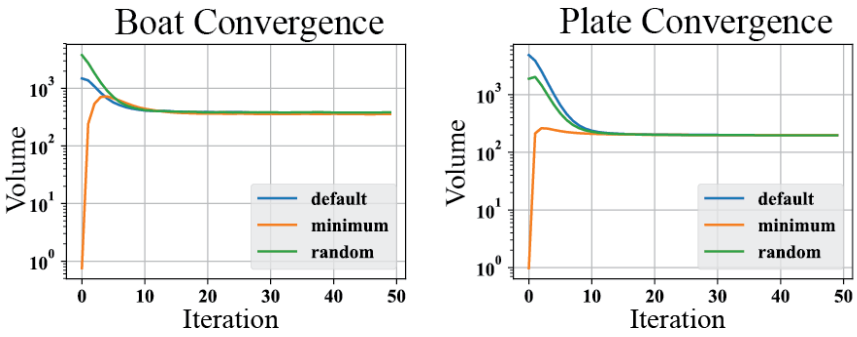


Figure 3.18: Left: Volume convergence for our method, for the boat model, using different starting points. Right: similar plots for the bending plate.

this initial solution by much after a few hundred iterations; although moving in the right direction, in terms of values, it may differ by a factor up to two from the optimal solution.

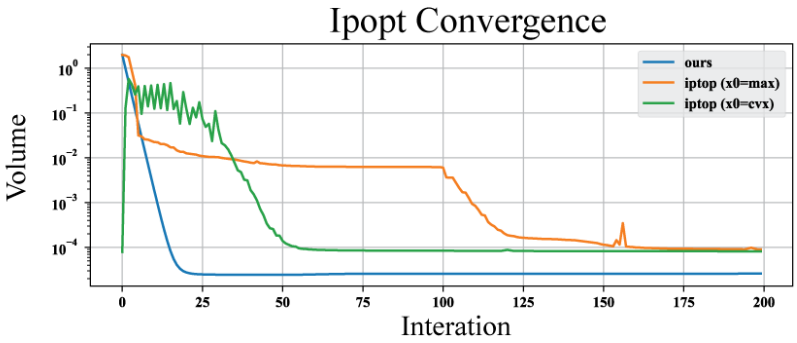


Figure 3.19: Volume vs. iteration step for a small cantilever example (408 beams) using Ipopt vs. our method.

While the SLSQP solutions exhibited oscillatory behavior, alternating decreasing the functional with decreasing constraints violations, the interior point method solutions mostly stayed close to the initial values. Figure 3.19 shows comparative results with Ipopt for a small cantilever test case. We observe that Ipopt converges to a volume almost an order of magnitude larger. When Ipopt is initialized with the solution of the convex problem with simplified volume functional (green), the optimization fails to find a better solution. Using a different initialization (orange: maximum width and thickness) does not provide better results. In contrast, our method converges to a much better solution in only a few iterations.

Effects of constraints on structure parameters. Figure 3.20 shows the effect of increasing maximal allowable thickness in a bending scenario, with the structure moving from fully solid, to a structure of narrow but tall beams (if the bound is increased to infinity, in principle, any bending force can be realized by a zero-volume infinitely thin rib).

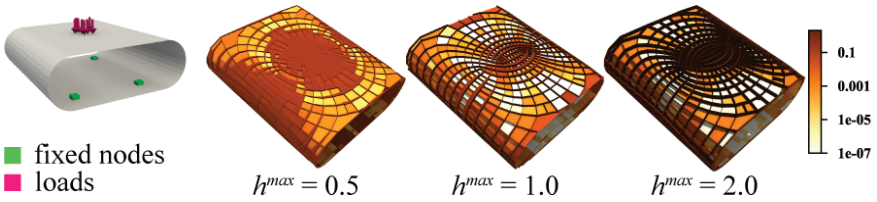


Figure 3.20: In a bending dominated structure, changing the maximal allowed thickness from small to large causes the results to change from a solid plate to a rib-like structure.

Similarly, Figure 3.21 shows the effect of increasing *minimal* thickness in a *tensile/compression* scenario. In this case, tall beams appear when increasing the lower bound because widths decrease to compensate for the increase of thickness. The usage of beams in tensile scenarios, while common, is often a consequence of constraints on minimum element size and are sub-optimal in the absence of these constraints. On the torsion cylinder experiment, the effect of using a minimal thickness of 0.04 vs. 0.00 increases the volume by 60%.

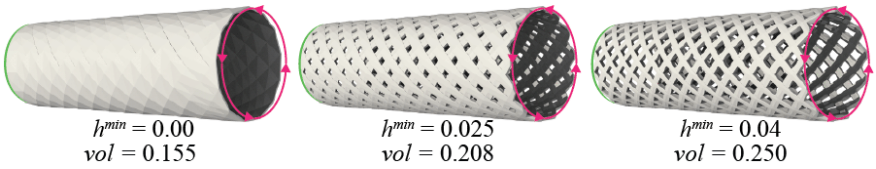


Figure 3.21: In a tensile dominated structure, changing the thickness lower bound from small to large causes the results to change from a solid plate to a rib-like structure. This example does not use a support shell ($h_s = 0.00$).

Relation to related work. A direct apples-to-apples comparison of different methods for optimization for shells is extremely difficult for a number of reasons, most importantly, because of different models used (e.g., all approaches use a variation of a beam model, and the specific choice has a significant impact on stress estimation). Another important reason

includes a strong dependence of the results on the maximal width/thickness/aspect ratio constraints. We summarize qualitative differences from the most closely related works Kilian et al. [49] and Li et al. [57].

The key difference from Kilian et al. [49] is that it changes the shape of the surface, focuses on the case when bending forces can be neglected, and uses the narrow-beam volume approximation. This may be appropriate for targeted architectural applications but different types of solutions, produced by our method, may be more optimal when these can be practically manufactured. In comparison, we deal with a shell of fixed surface shape, take bending forces into account and use a more precise volume approximation, which leads to a non-convex problem.

The method of Li et al. [57] is closer to ours: they construct a quadrangulation following a field to determine beam directions, and also take bending into account in the elasticity model. However, rather than using an optimized stress field as we do, they use the stress field of the non-optimized shell. Although the fields are typically close, in some cases the optimized field provides an advantage (Figure 3.17).

More importantly, Li et al. [57] uses a narrow-beam approximation for the volume which has a major effect in the efficiency of the reinforced structure. To measure this effect, we compare the results of our optimization using different volume formulas as functional (Figure 3.22). The results show that using the simpler volume approximation can increase the volume used by up to 44%.

As explained on Section 3.3, the convex volume approximation biases the solution against using variable-thickness shells. To demonstrate the effects of this, we compare the stiffness of two reinforcements, obtained using different methods, with the same material cost. The shell and rib-reinforcement structure (Figure 3.23b) where provided by the authors of Li et al. [57], while the second type of reinforcement (Figure 3.23a) was generated with our algorithm. To compare the results fairly, with no dependence on the beam model used, we

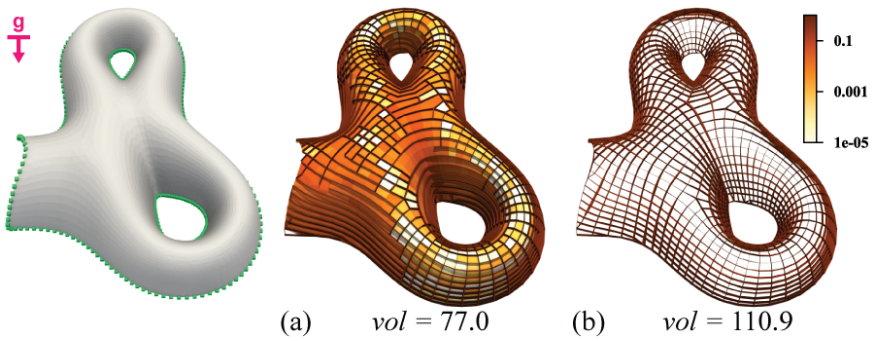


Figure 3.22: Optimization results using different volume formula: (a) exact volume, (b) approximated convex volume.

post-process both results using TetWild meshing software [44] to create tetrahedral meshes of similar density, and then use Finite Element Analysis to solve for the displacements with the loads and supports used in optimization (we use the PolyFem implementation [80]). We impose zero-displacements on the nodes at floor level, a uniform vertical load on the rest of the structure, a Young's modulus $E = 210000$ and Poisson's ratio $\nu = 0.3$. The resulting displacements are shown on Figure 3.23c, d. Our reinforced structure shows a maximum displacement of 67.31, while the rib structure of Li et al. [57] shows a higher maximum of 97.16. Thus, under the same boundary conditions, our result achieves 1.44 times higher strength compared to a rib-reinforced structure of same weight.

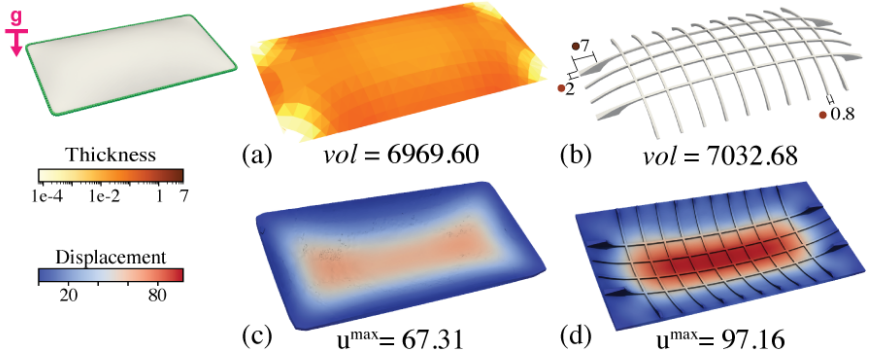


Figure 3.23: Finite Element Analysis of reinforced structures: (a) variable thickness reinforcement generated by our system; (b) rib-reinforcement from Li et al. [57]; (c,d) simulated displacements on reinforced structures.

This comparison does not include the I-beam profile optimization of beam cross-sections introduced by Li et al. [57] as I-beam shapes were not included in the geometry provided by the authors and relevant code is no longer available. However, in the case of dominant tensile/compression loads along the beam, this technique is not likely to yield a significant improvement, as these forces depend primarily on the cross-section area.

Physical Experiments. We have printed several simple structures to validate our optimization experimentally (Figures 3.24, 3.25, 3.26). Due to the highly approximate nature of the physical model used, we did not attempt a quantitative match to the simulated values, but we did closely match the values of the printed models. The comparison in all cases is between an optimized model and a uniform-thickness model of the same weight. Observed displacements in all cases differed by a factor more than 3, suggesting a similar difference in stress.

The optimization of the shelf was done using distributed loads (Figure 3.28, last row). However, when the shelf bends, the physical load is applied in only two separated regions. Using properly distributed loads would, most likely, only increase the difference in performance, as the structure was optimized for this case.

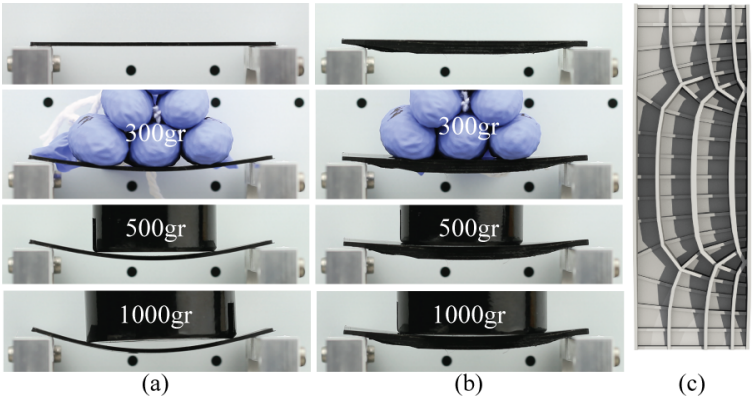


Figure 3.24: (a) shelf of uniform thickness; (b) optimized shelf of same weight; (c) model of optimized shelf.

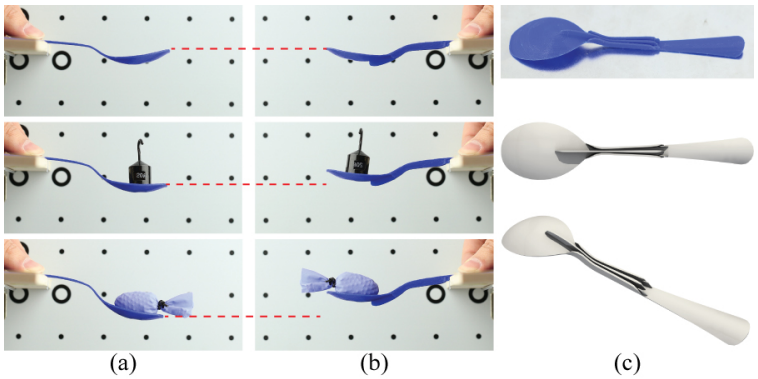


Figure 3.25: (a) spoon of uniform thickness; (b) optimized spoon of same weight; (c) model of optimized spoon.

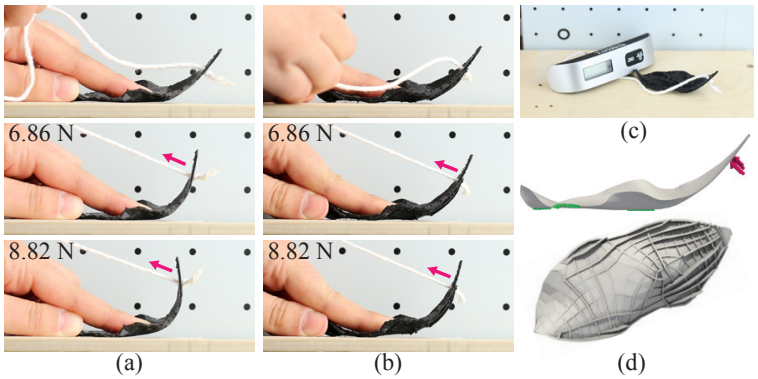


Figure 3.26: (a) leaf of uniform thickness; (b) optimized leaf of same weight; (c) we use a digital scale to measure the pulling force; (d) model of optimized leaf.

Finally, Figures 3.27 and 3.28 show a set of optimized shells obtained for a variety of shapes using our method; in all cases we have preserved a minimal width/thickness beam to indicate the mesh edges, but the load is carried by a relatively small number of beams; we found that ribs tend to appear, even in tension areas, unless the thickness bound is set to very low values. This is consistent with the observation that with no thickness, the load carried by bending forces is maximized, if the goal is to reduce the volume.

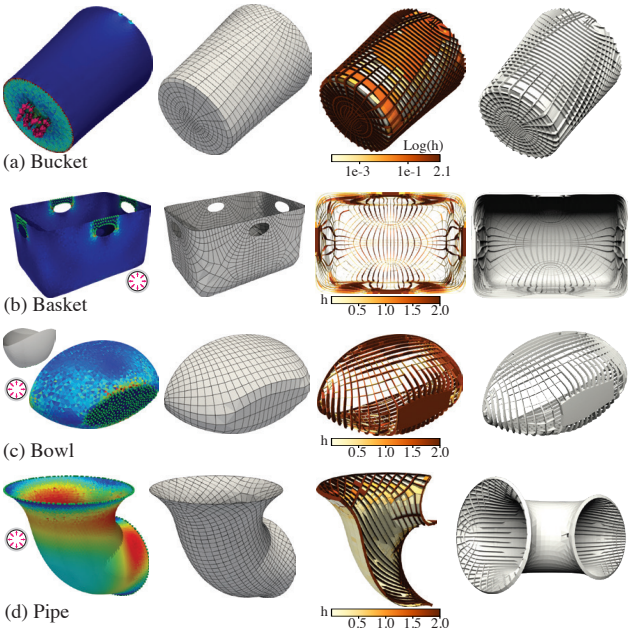


Figure 3.27: Examples of consumer product objects. For each one, images show loads and initial stress distribution, quadrangulation, cell optimization colored by thickness, and final geometry

3.9 Conclusions and Future Work

In this chapter, we have described a way to approximate and efficiently solve the problem of minimizing the weight of a support structure for a shell. Our work focuses on choosing a simple, yet sufficiently expressive, geometric description of the support structure, and an optimization algorithm capable of optimizing it. The proposed method separates the construction into three stages, with the first stage optimizing the field the beam directions must follow and creating a corresponding quad-dominant mesh, the second stage creating a cell structure with optimized shape parameters, and the third stage creating an actual realization.

This makes our approach particularly flexible and allows to integrate a variety of additional user inputs and constraints, e.g., by modifying the field to change the truss directions, or by adding beams in the second stage to support a connection to a separate object. It is also

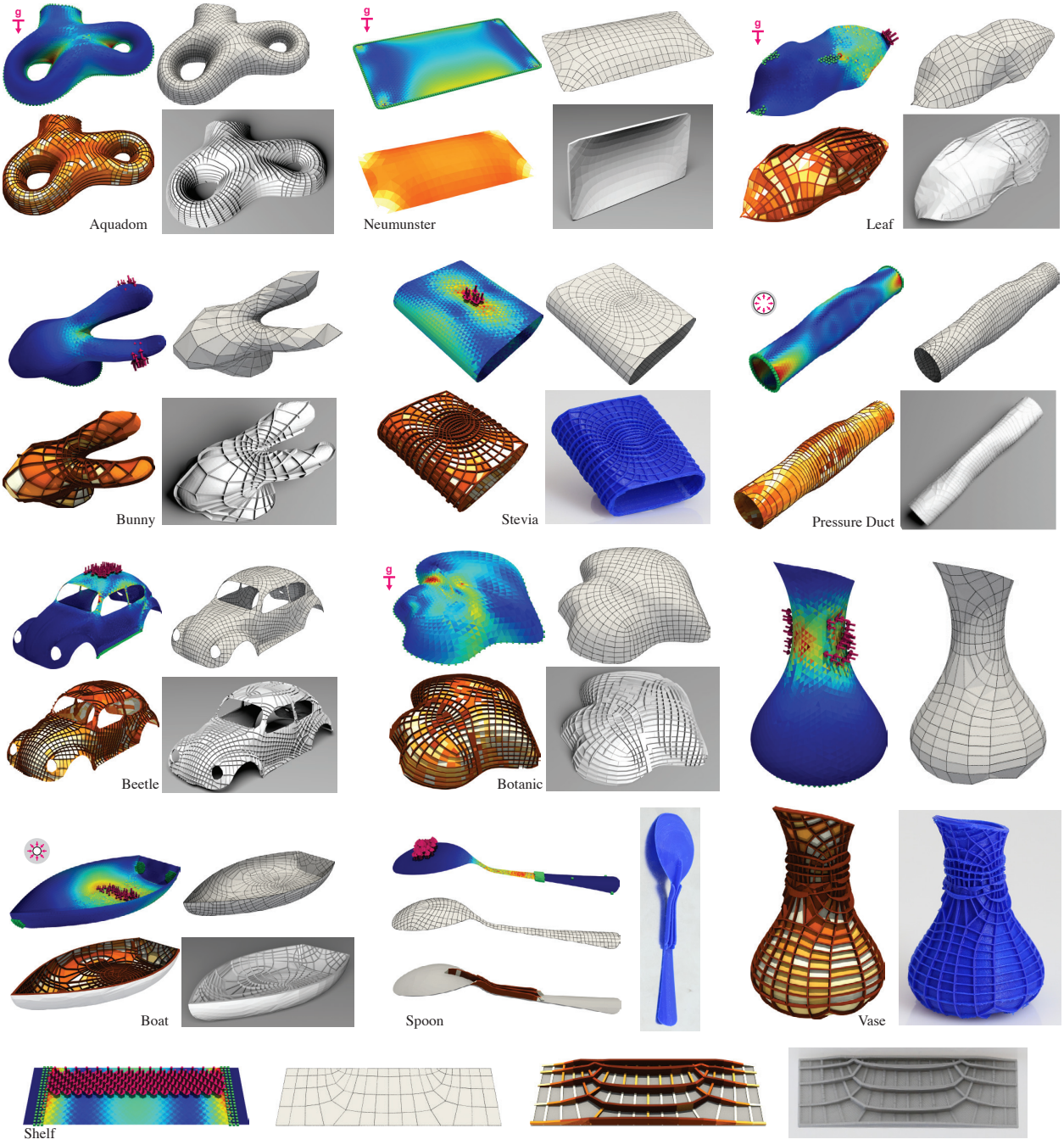


Figure 3.28: Examples of structures obtained for a variety of shapes and loads. For each one, images show loads and initial stress distribution, quadrangulation, cell optimization (colored by thickness in logarithmic scale) and final geometry.

	d(mm)	$\|F\|$	h	h^{s}	k	s	$t_{\mathrm{solve}}(s)$
Aquadom	85.60	7742	0.5	0.005	100	0.2	72.330
Basket	127.56	10943	2.0	0.06	30	0.5	72.513
Beetle	169.01	7558	1.2	0.012	50	0.5	35.999
Boat	92.60	9084	1.0	0.05	100	0.2	33.067
Botanic	42.59	2152	0.5	0.01	100	0.5	35.586
Bowl	117.77	4779	2.0	0.0	32	0.2	12.354
Bucket	150.96	7860	2.1	0.21	50	0.5	45.744
Bunny	106.00	7471	2.0	0.9	100	0.25	22.291
Duct	460.58	6932	1.0	0.01	100	0.2	69.593
Leaf	99.24	2980	2.0	0.9	46	0.5	1.954
Neumunster	111.92	6072	2.0	1.2	63	0.2	6.959
Pipe	159.48	10409	2.0	0.0	82	0.2	80.226
Shelf	88.54	2400	4.0	0.5	60	0.5	1.663
Spoon	133.11	6528	4.0	0.9	29	0.5	4.129
Stevia	90.68	4840	2.0	0.9	50	0.5	23.941
Vase	69.64	4798	1.0	0.45	100	0.5	21.817

Table 3.1: Statistics for the 3D models in Figure 3.28 and Figure 3.1: the bounding box diagonal d (mm), the number of triangles $|F|$ in the input mesh, the maximum thickness h (mm) of the beams, the constant shell thickness h^{s} (mm), the number of iterations k and step-size s of the optimization solve, and the time t_{solve} it took to complete all iterations.

very efficient, with optimization converging in a few iterations, and quite consistently.

While we use a highly simplified model for cell mechanics, the overall approach admits replacing this model with a more advanced finite element formulation. In this case, it is likely that the local step would require numerical optimization; however, as long as the model for a cell stays low-parametric, one is likely to be able to solve it efficiently.

Future Work. In the future, we will study the effect of alternating optimizing the cell mesh $\mathcal{P}$, which defines the layout of the beams, and optimizing beam parameters. While our evaluations on the role of beam directions (Figure 3.17) suggest only an incremental benefit, it might help regions with large stress concentrations. Multiple loading scenarios can be optimized for simultaneously, following the approach of [87]), i.e., computing stresses for each load case and constraining them to be bounded.

	$V^s(mm^3)$	$V^b(mm^3)$	$h_{eq}(mm)$	c_{eq}/c
Aquadom	16.989	90.335	0.032	1.601
Basket	1207.584	3392.022	0.229	7.983
Beetle	170.128	1287.227	0.103	5.739
Boat	189.997	418.539	0.160	7.099
Botanic	9.134	80.533	0.098	1.266
Bowl	0.000	4083.710	0.410	4.437
Bucket	105.195	58.274	0.326	26.924
Bunny	7270.428	1134.806	1.040	1.229
Duct	1075.645	19777.817	0.194	2.105
Leaf	2304.430	659.348	1.158	1.105
Neumunster	44378.721	3902.169	9.139	1.076
Pipe	0.000	45833.029	2.551	1.161
Shelf	1176.000	1386.550	1.090	7.252
Spoon	1544.960	326.733	1.090	18.621
Stevia	8329.664	2568.409	1.178	1.198
Vase	3505.556	492.212	1.026	1.953

Table 3.2: Performance for the 3D models in Figure 3.28. We show the ratio c_{eq}/c between the compliance c of our results and the compliance c_{eq} of a shell of constant thickness of the same weight. The fixed shell volume V^s, the support structure volume V^b, the thickness h_{eq} of the equivalent-weight constant thickness shell.

Limitations. Our work has two main limitations: first, the formulation for the field optimization still has restrictive low-volume and fixed-thickness assumptions. Based on our evaluation of field direction sensitivity of the final design, we do not view this as a major limitation. The second, more significant, limitation is the highly simplified model we used. If exact results are needed, this model can be used to quickly obtain an initial result, which can then be refined using a more advanced mechanical description of cells and shape optimization.

3.10 Chapter Notes

3.10.1 Convexity of truss continuum optimization

Expressing the functional in terms of entries of the stress matrix, using $a = (\sigma_{11} + \sigma_{22})/2$, $b = (\sigma_{11} - \sigma_{22})/2$, and $c = \sigma_{12}$, we obtain

$$|\lambda_1(\sigma)| + |\lambda_2(\sigma)| = 2\max(a, \sqrt{b^2 + c^2}) \tag{3.19}$$

which verifies convexity of the energy.

3.10.2 Dual of the continuum shell problem

This derivation follows [90], which we include here for completeness.

$$L(\sigma, \mathbf{u}) = \frac{E}{\sigma_0} \int_\Omega |\lambda_1(\sigma)| + |\lambda_2(\sigma)| dA + \int_\Omega \mathbf{u}^T \mathrm{div}\sigma dA - \int_\Omega \mathbf{f}^T \mu dA.$$

Integrating by parts, and assuming either free boundary $\partial_n \mathbf{u} = 0$ or fixed boundary $\mathbf{u} = 0$, we obtain

$$L(\sigma, \mathbf{u}) = \frac{E}{\sigma_0} \int_\Omega |\lambda_1(\sigma)| + |\lambda_2(\sigma)| dA - \int_\Omega \varepsilon(\mathbf{u}) : \sigma dA - \int_\Omega \mathbf{f}^T \mathbf{u} dA.$$

To obtain the dual we need to minimize over all possible σ in a coordinate system aligned with ε the expression

$$\frac{E}{\sigma_0}(|\lambda_1(\sigma)| + |\lambda_2(\sigma)|) - \lambda_1(\varepsilon)\sigma_{11} - \lambda_2(\varepsilon)\sigma_{22}.$$

The minimum of this expression is $-\infty$, if $|\lambda_i(\varepsilon)| > 1$, for either $i = 1$ or $i = 2$; otherwise, it is zero; this leads to the dual problem (3.10).

3.10.3 Proof of the properties of $V(z)$.

We drop the cell superscript c in this proof. We assume that $z_1 \leq z_2 \leq z_3$. By the constraints of the problem, $y_1 + y_2 + y_3 \leq 1$, $0 < z_{min} \leq z_i$.

$V(y, h)$ (3.1) is a quadratic function of y_i and linear in h_i; once we express it in terms of z_i, to eliminate the constraint between h_i and z_i, it becomes a cubic function of z_i, with Hessian

$$H(z) = \begin{bmatrix} A & -k_1 g_2^b & -k_1 g_3^b \\ -k_1 g_2^b & B & -k_2 g_3^b \\ -k_1 g_3^b & -k_2 g_3^b & -1/2 g_3^b \left(g_3^t + 3k_3\right) \end{bmatrix}$$

where

$$k_i = g_i^t + 2 g_i^b z_i = dw_i/dz_i,$$

$$B = -1/2 g_2^b \left(g_2^t + 3k_2 + 2g_3^t + 2k_3\right),$$

$$A = -1/2 g_1^b \left(g_1^t + 3k_1 + 2g_2^t + 2k_2 + 2g_3^t + 2k_3\right).$$

A direct evaluation shows that $v^T H(z) v$ evaluated for $v = [1, 0, 0]$ is negative for positive z_i and $g_1^b \neq 0$. Similar is true for $v = [0, 1, 0]$ and $v = [0, 0, 1]$. We conclude if $g_i^b \neq 0$ for some i, any critical point in the interior of the domain is a maximum or a saddle, so there are no minima in the interior. If all g_i^b are zero, the volume is a linear function of z_i, and the optimum is also on the boundary.

The constraint $y_1 + y_2 + y_3 \leq 1$ defines, if $g_b^i \neq 0$ for some i a quadratic surface in z_i coordinates, and the constraints $z_1 \leq z_2$, $z_2 \leq z_3$, $z_1 \geq z_{min}$ define 3 halfspaces. Their intersection is a tetrahedron with a curved face on the ellipsoid. The faces of the feasible domain correspond to one of the inequality constraints becoming an equality. Similarly, by a direct calculation, substituting either $z_1 = z_{min}$, $z_2 = z_1$ or $z_2 = z_3$ into $V(z)$ and computing

2 Hessians, we observe that these are not positive definite, hence there can be no solution on faces corresponding to the linear constraint. Finally, the same fact holds for pairs of constraints that define edges of the constraint domain, e.g., $z_1 = z_{min}$ and $z_2 = z_1$. We conclude that either the solution is on the face corresponding to the constraint $y_1 + y_2 + y_3 = 1$ or at the only vertex not on this face of the feasible domain, $z_i = z_{min}$, $i = 1, 2, 3$ (case 1 of the proposition). If the minimum satisfies $y_1 + y_2 + y_3 = 1$, it is either in the interior (case 2), or on one of the three edges of the boundary of that face (cases 3-5).

3.10.4 Discrete bending strain for beams

The common normal $\mathbf{n}_i$ at i is defined as

$$\mathbf{n}_i = \sum_{\ell \in N(i)} \mathbf{e}_{i\ell} \times \mathbf{e}_{i\, next(\ell)}$$

where $next(\ell)$ is the edge following ℓ in CCW order around i.

At each vertex, we define a normal $\mathbf{n}_i$, as the average of cross-products of pairs of incident edges. For an edge $\mathbf{e}_{il}$, incident at a vertex i, let $\mathbf{e}_{im}$ be the next CCW edge, and $\mathbf{e}_{ip}$ be the previous edge. We define

$$\mathbf{q}_{lm} = \mathbf{e}_{il} \times \mathbf{e}_{im}$$

the (unscaled) normal to the triangle formed by i, l, m. Then $\mathbf{n}_i = \sum_{l,m \in N(i)} \mathbf{q}_{lm}$, and $\hat{\mathbf{n}}_i = \mathbf{n}_i / |\mathbf{n}_i|$.

We express the bending strain in the form $D^b_{ij} u$, where u is the vector of all vertex displacements. Define $\Delta \mathbf{e}_{ij} = \mathbf{u}_j - \mathbf{u}_i$. Then $\Delta \mathbf{q}_{lm} = \Delta \mathbf{e}_{il} \times \mathbf{e}_{im} + \mathbf{e}_{il} \times \Delta \mathbf{e}_{im}$. The part of $\Delta \mathbf{q}_{lm}$ affecting $\Delta \hat{\mathbf{n}}_i$ is the part perpendicular to $\hat{\mathbf{n}}_i$:

$\Delta q^{\perp}_{lm} = (I - \hat{\mathbf{n}}_i \hat{\mathbf{n}}_i^T) \Delta \mathbf{q}_{lm} = P_i \Delta \mathbf{q}_{lm}$, where P_i is the matrix $(I - \hat{\mathbf{n}}_i \hat{\mathbf{n}}_i^T)$. By substitution

into the expression for n we get:

$$\Delta\hat{\mathbf{n}}_i = \frac{1}{|\mathbf{n}_i|} P_i \sum_{l \in N(i)} g_l \times (\mathbf{u}_l - \mathbf{u}_i)$$

where g_l is defined as follows:

- $\mathbf{g}_l = \mathbf{e}_{ip} - \mathbf{e}_{im}$, if both p,m are in $N(i)$, i.e. $\mathbf{e}_{il}$ is not on the boundary.
- $\mathbf{g}_l = \mathbf{e}_{ip}$, if $m \notin N(i)$.
- $\mathbf{g}_l = -\mathbf{e}_{im}$, if $p \notin N(i)$.

For a vector $\mathbf{a} = [a_x, a_y, a_z]$, let $R(a)$ be the infinitesimal rotation matrix about a:

$$\begin{pmatrix} 0 & -a_z & a_y \\ a_z & 0 & -a_x \\ -a_y & a_x & 0 \end{pmatrix}.$$

Then

$$\Delta\hat{\mathbf{n}}_i = \sum_{l \in N(i)} M_l^i (\mathbf{u}_l - \mathbf{u}_i)$$

where $M_l^i = \frac{1}{|\mathbf{n}_i|} P_i R(\mathbf{g}_l)$ is a 3×3 matrix.

$$\varepsilon^b[ij] = h \left(\sum_{l \in N(j)} (\mathbf{d}_{ji}^l)^T (\mathbf{u}_l - \mathbf{u}_j) + \sum_{l \in N(i)} (\mathbf{d}_{ij}^l)^T (\mathbf{u}_l - \mathbf{u}_i) \right)$$

where $\mathbf{d}_{ij}^l = -(M_l^i)^T \hat{\mathbf{e}}_{ij} / l_{ij}$, a vector of length 3. From this expression, we can immediately obtain D_{ij}.

Chapter 4

Robust Collision Resolution

This chapter is based on unpublished work in collaboration with co-authors Zachary Ferguson, Teseo Schneider, Danny Kaufman, Daniele Panozzo, and Denis Zorin. My contributions include the development of the rigid body simulation, implementation of the distance barrier solver, performance and stability analysis of our method, and comparisons with Lu et al. [59].

4.1 Introduction

This project was motivated by the observations we have made when experimenting with optimization of packaging constructed from folded chapter or cardboard. The overall goal was to study design optimization for scenarios involving rapid acceleration and deceleration. In our prototype system (Figure 4.1), which included a basic 2D deformable sheet simulation, with a collision resolution system based on the work of [59], we have observed that collision robustness is critical for being able to use the simulation in the optimization loop. While the algorithm worked for select configurations, matching experimental results, it failed for other configurations, not resolving collisions correctly. In addition, to our own simulator,

we simulated the same scenarios in Box2D, a widely used 2D simulation engine, making the same conclusions: When exploring a large design space, especially involving fully folded sheets, simulation would inevitably fail in some instances. To address this problem, we have developed a new collision-resolution method aiming to achieve maximal robustness, as independent as possible from the parameters of the simulation.

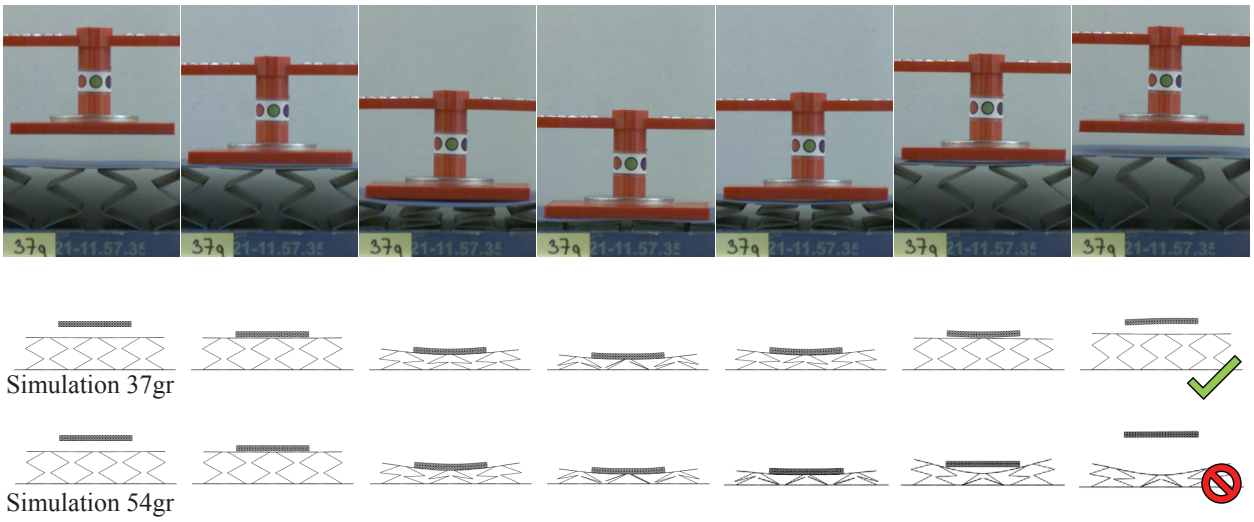


Figure 4.1: Top row: free fall experiment on top of chapter cushion, recorded at 4k frames per second; middle row: simulation of the same scene using a 37gr weight; bottom row: simulation using a 57gr weight fails due to interpenetrations not solved correctly.

More generally, collision detection and response algorithms are essential for physical modeling with applications in many disciplines including robotics, computer graphics, mechanical engineering, and scientific computing. These algorithms have been an active research topic for decades, with methods falling broadly into two categories: accurate methods, which usually have a high computational cost, and less accurate methods, which focus on real-time performance. We note that accuracy is not necessarily correlated with robustness, and accurate algorithms, may, e.g., place strong restrictions on the time step magnitude.

In both cases, these algorithms rely on user-specified parameters and often become unstable, slow, or produce unrealistic results if wrong choices are made. While this is not a problem

for many applications in entertainment and engineering, where per-simulation parameter tuning is tedious but feasible, this state of affairs is problematic in applications requiring to batch process many simulations. For example, the computation of shape derivatives for inverse design or the generation of large training sets for machine learning: in these cases, it is impossible to manually tune the simulation parameters at every run, due to both the impractical time investment and due to the fact that changing parameters will likely affect the simulation results, making them hard to compare.

We aim to develop a formulation tailored for applications when fully automatic handling of collisions with no parameter tuning is essential, such as optimization. Our formulation is based on formulating the problem as an inequality constrained system of equations, which we solve using a customized feasible interior point method. This methods assume a valid, collision-free initial state, and keep the validity as an invariant during the entire simulation. Two main features distinguish our solver from standard barrier solvers: (1) we use a locally supported barrier function, so only a small percentage of barriers need to be explicitly included during the solve, and (2) we implement an intersection-preventing line search, based on an exact predicate, to ensure that the state remains valid at the inner iterations.

We demonstrate the robustness and practical applicability of our method in a set of challenging simulation cases, ranging from large displacements on small scenes (high-speed collisions), to long interlocking chains composed of rigid blocks (multiple simultaneous collisions). We compare our technique to existing state of the art methods for a wide range of parameters, demonstrating that our method is the only one supporting batch processing of simulation scenarios involving rigid bodies.

4.2 Related Work

The focus of the work on contact has been divided between methods that aim for efficiency (e.g. real-time simulations) and those that strive for accuracy and robustness. We are specially interested in methods that resolve interpenetration in full, and that are robust under different simulation parameters.

Most algorithms for contact response can be categorized as either penalty methods, impulse-based methods, or constraint-based solvers.

Penalty methods detect when objects are about to collide and apply a repulsive force to prevent penetration. As distance between objects decreases, the force is also increased. These methods are easy to implement but the system becomes stiffer for large forces, and they cannot ensure collisions are resolved. On the other hand, impulse-based methods attempt to guarantee no inter-penetrations are produced. Starting from a collision-free state at time t_0, collisions that occur between t_0 and $t_0 + \Delta t$ are resolved applying an impulse that *instantaneously* corrects the velocities. These approaches cannot guarantee success in cases with multiple conflicting constraints, and may lead to jittering.

In contrast to the previous two methods, constraint-based methods are often more robust. These methods formulate the no-interpenetration problem as a general constrained optimization problem. The main differences between the approaches lay in (1) the discretization of the functional (equation of motion), (2) the choice of constraint or gap-function, (3) and the solver used to solve the system of equations.

Non-interpenetration constraints These constraints are often *gap functions*, where a positive value indicates no interpenetration, a negative means interpenetration, and zero indicates contact. While the nature of the non-penetration constraints is usually non-linear, the majority of previous work linearizes the constraints using e.g. half-plane constraints [89],

acceleration-based [13], velocity-based [48],and position-based constraints [70], and linearized interpenetration volume [6]. Linearized constraints lead to Linear Complementarity Problems (from the KKT system of equations), which are typically solved using iterative solvers. However, using a linearized approximations of the constraints often results in interpenetration. A number of methods were proposed to reduce or eliminate this problem, typically requiring additional loops to improve the linearization or refine the constraints. For example, [70], after computing a step with linearized constraints, they check for collisions using CCD and, if collisions are found, they iteratively refine the constraint manifold.

Gap functions based on interpenetration volumes or signed distances (e.g. [6]) have the additional problem of not handling sheets robustly: these gap functions can be used only for shapes with an interior. To eliminate this problem, [38] and [59] use the space-time volume instead, however the gradient of this volume is zero in the feasible domain, making it difficult to use standard solvers. In [59], the resulting NCP problem is solved using minimal-map Newton method. Another approach to handling thin objects is proposed by Müller et al. [65], where they triangulate the "air" surrounding the objects and enforce triangles do not flip during simulation. This approach requires extensive remeshing on iterations with large displacement or rotations, making the method too expensive.

Distance-based gap functions are another approach that works well for closed and open shapes. In [39], a modified penalty method is used, with discrete penalty layers based on the distance between two points. Additionally, this work uses a contact potential that grows to infinity as the points near a given surface thickness. Nested penalty layers require using an adaptive time integrator, which can lead to expensive time-stepping. In Kaldor et al. [47], to prevent collisions between yarn threads, a barrier function based on the distance between two threads is added to the energy, which pushes threads apart. Both methods suffer from the limitations associated with penalty methods, and cannot ensure the final state is collision-free.

Interior Point Methods To be able to work with thin shells, as well as solid objects, our method uses the unsigned distance as a gap function, with a minimal separation distance required (however, it can be arbitrarily small). To guarantee that we end in a collision-free configuration, we use a barrier-based interior point method which, starting from a feasible point, always reminds in the feasible domain.

Interior point methods are widely used for constrained optimization problems [20]. We can roughly classify these methods into three subcategories: (1) those that starting from a feasible point always stay in the feasible domain; (2) those that start with an unfeasible point, and have a first stage where they find a feasible one, and then always stay in the feasible domain; and (3) methods that take some steps in the unfeasible domain. The latter allows for faster convergence, but cannot guarantee the constraints are satisfied either at the intermediate steps, or in the final configuration. As the overall problem is non-convex, it is not guaranteed that the solver will find a feasible point, once a configuration becomes infeasible (except through complex backtracking). In the case of collisions, maintaining feasibility at all times ensures that this type of problems is avoided. Our solver is based on a standard barrier method extended to work robustly with collisions (Section 4.5).

Commercial Physics Engines While there exists a great number of commercial physical engines, most of them are design to work only in 3D. For 2D rigid body simulation, Box2D is the most widely used physics engine [24]. It features continuous collision detection, friction, and restitution. The contact constraints are solved using sequential impulses, and a Semi-Implicit Euler time integration is used for time-stepping [25]. Because of design decisions, Box2D only handles a limited set of shapes for dynamic rigid bodies. Namely, Box2D only supports circles and convex polygon (of up to eight vertices by default). However, one can compound these shapes to create complex bodies. In contrast, our simulator can support any shape with a piece-wise linear boundary (closed or open) but not circles. We compare

the robustness of our method against Box2D on Section 4.6. For a fair comparison, we use compound shapes compatible with Box2D. However, in our simulation, we only include the boundary edges needed for collision detection.

4.3 Constrained Rigid Body Dynamics

We use a standard model for 2D rigid body motion. We consider a collection of n rigid bodies with generalized coordinates $\mathbf{q} = (x, y, \theta) \in \mathbb{R}^{3n}$, where (x, y) is the spatial position and θ is the rotation angle. The generalized velocities are $\mathbf{v} = (v_x, v_y, \omega) \in \mathbb{R}^{3n}$, where ω is the angular velocity. The mass matrices $\mathbf{M} \in \mathbb{R}^{3n \times 3n}$ are diagonal matrices with diagonals (m, m, I), where m is the object mass and I is the moment of inertia. Finally, the vector of external forces $\mathbf{F}^e = (F_x^e, F_y^e, \tau) \in \mathbb{R}^{3n}$, has torque τ as its last component.

For an arbitrary point $\mathbf{r}_i^0$ on a rigid body in *body space*, the point's position in world space is given by

$$\mathbf{r}_i = \mathbf{R}(\theta)\mathbf{r}_i^0 + \mathbf{X} \tag{4.1}$$

where $\mathbf{R}$ is a 2×2 rotation matrix, θ is the rotation angle of the rigid body, and $\mathbf{X}$ is the position of the rigid body's center of mass. Similarly, the function $\mathbf{f} : \mathbb{R}^{3n} \to \mathbb{R}^{2n}$ transforms the rigid bodies' pchapters from local space to world space, such that $\mathbf{x} = \mathbf{f}(\mathbf{q})$ are the vertex positions.

Equations of motion. In these variables, the equations of motion are given by

$$\mathbf{M}\ddot{\mathbf{q}} = \mathbf{F}^e(\mathbf{q}, \dot{\mathbf{q}}) + \mathbf{F}^c(\mathbf{q}) \tag{4.2}$$

where $\mathbf{F}^c(\cdot)$ are additional contact forces, which are delta functions corresponding to instant changes in velocities. (Currently we do not consider friction, which does need to be added,

and will require substantial changes).

Constraints Additionally, we impose an explicit no-interpenetration constraint.

These constraints are often imposed using a signed distance or another type of *gap function*, that has negative values if two objects interpenetrate, positive values if the objects are separated, and zero if they are in contact. This type of constraints requires the assumption that the objects have an interior, which is restrictive for modeling thin sheets and one-dimensional strands.

We use a formulation that applies in all cases, by adding a small expansion to all objects, i.e., if the distance between two points is exactly δ_{min}, these points are in contact, and if it is less, then there is interference. In other words, we use $d(\mathbf{q}) - \delta_{min}$ as the gap function, where d is the distance between actual objects. It also has the advantage of being defined, e.g., for pairs of points or a point and a segment: interpenetration for points corresponds to an overlap between two disks of radius $\delta_{min}/2$ centered at these points. The downside of using a function like this in the context of some numerical methods is that it not smooth for $\mathbf{q}$ for which $d(\mathbf{q}) = 0$, thus methods allowing infeasible intermediate values may not perform well. We use a solver keeping iterations in the feasible domain, which fortunately coincides our goal of robustness (e.g., even if we terminate early, the solution may be inaccurate, but is still valid).

With these assumptions, the constraint can be formulated in the following form:

$$d(\mathbf{r}(\mathbf{q}), \mathbf{r}'(\mathbf{q})) - \delta_{min} \geq 0, \text{ for all distinct point pairs } \mathbf{r}, \mathbf{r}' \text{ from different objects} \tag{4.3}$$

We note that compared to commonly used formulations, this one is more abstract, as it involves an uncountable set of constraints parametrized by $\mathbf{r}, \mathbf{r}'$. E.g., we do not assume an active set of contacts tracked in the formulation itself, or use distances between discrete rigid

objects. It is useful to view these aspects of the final algorithm as a part of discretization and acceleration. We also note that this constraint needs to be altered for deformable objects which may have self-collisions.

Collision response. We use impulse-based Newton model for rigid body collisions (e.g., Baraff [14]). Suppose A and B are two colliding bodies, and let $\mathbf{r}_A$, resp. $\mathbf{r}_B$ be the vectors from the center of mass of A, resp. B to the first impact point. The velocities of the collision points before collision are

$$\begin{aligned} \dot{\mathbf{r}}_A &= \omega_A \mathbf{r}_A^{\perp} + \mathbf{V}_A, \\ \dot{\mathbf{r}}_B &= \omega_B \mathbf{r}_B^{\perp} + \mathbf{V}_B, \end{aligned} \tag{4.4}$$

where $\perp$ denotes a counter-clockwise rotation, $\mathbf{V}_{A,B}$ is the linear velocities of the center of mass, and $\omega_{A,B}$ the angular velocities.

The change of velocity after collision is due to an impulse (i.e, delta-function force), applied along the normal direction at the collision point. This direction is unambigously defined for two smooth surfaces in contact, but requires more effort to define if the objects have sharp features, which is common; the choice of the normal direction is also important. In two dimensions, if $\mathbf{r}_A$ is point of normal discontinuity on A, and $\mathbf{r}_B$ is a smooth point, the normal of A is used. In the degenerate case of vertex-vertex collision, the vector connecting $\mathbf{r}_A$ and $\mathbf{r}_B$ is a suitable choice. For shapes enclosing a nonempty volume, the normal direction points in the outward direction of the body. For shapes with empty interior we define the "outward" direction to be pointing towards the direction from which A is coming, i.e., so that the relative velocity for approaching objects is positive.

Now we can define the relative velocity of the collision points as:

$$\dot{\mathbf{r}}_{rel} = (\dot{\mathbf{r}}_A - \dot{\mathbf{r}}_B) \cdot \mathbf{n} \tag{4.5}$$

For friction-less collisions, the restitution law states $\dot{\mathbf{r}}^+_{rel} = -C\dot{\mathbf{r}}^-_{rel}$, where the superscript $-$ and $+$ are used to indicate the quantities before and after applying the impulse. The coefficient of restitution, $C \in [0, 1]$, relates to the elasticity of the collision, such that $C = 0$ corresponds to a perfectly inelastic collision, and $C = 1$ to one perfectly elastic.

The change in the velocity of the center of mass due to an impulse of magnitude j acting on A, and magnitude $-j$ acting on B are, by 3rd Newton's law,

$$\mathbf{V}^+_A - \mathbf{V}^-_A = m_A^{-1} j\mathbf{n}, \quad \mathbf{V}^+_B - \mathbf{V}^-_B = -m_B^{-1} j\mathbf{n}. \tag{4.6}$$

where m_A, resp. m_B is the mass of A, resp. B. Similarly, the change in angular velocity is

$$\omega^+_A - \omega^-_A = I_A^{-1} j(\mathbf{n} \cdot \mathbf{r}^\perp_A), \quad \omega^+_B - \omega^-_B = -I_B^{-1} j(\mathbf{n} \cdot \mathbf{r}^\perp_B) \tag{4.7}$$

where I_A, resp. I_B is the moment of inertia of A, resp. B, which are constant in 2D.

Using equations (4.6) and (4.7) we can expand the relative velocity change to

$$\dot{\mathbf{r}}^+_{rel} - \dot{\mathbf{r}}^-_{rel} = j\left((m_A^{-1} + m_B^{-1} + I_A^{-1}(\mathbf{n} \cdot \mathbf{r}^\perp_A)^2 + I_B^{-1}(\mathbf{n} \cdot \mathbf{r}^\perp_B)^2\right) = Kj \tag{4.8}$$

Finally, using $\dot{\mathbf{r}}^+_{rel} = \dot{\mathbf{r}}^-_{rel} + Kj = -C\dot{\mathbf{r}}^-_{rel}$, we obtain the following expression for the impulse magnitude:

$$j = -K^{-1}(1 + C)\dot{\mathbf{r}}^-_{rel}$$

4.4 Discretization

Each 2D rigid body is represented as a collection of segments that form a connected shape (although this is not essential) and can overlap only at vertices. Any number of segments can be connected at vertices

Discretization of the equations of motion. Given a time-step, with initial positions, $\mathbf{q_0}$, and initial velocities $\mathbf{v_0}$, we use Semi-Implicit Euler to compute positions and velocities at the end of the time-step. The discrete equations of motion, including contact forces $\mathbf{F}^c \in \mathbb{R}^{3n}$, are

$$\mathbf{v}_1 = \mathbf{v}_0 + h\mathbf{M}^{-1}\mathbf{F}_0^e + h\mathbf{M}^{-1}\mathbf{F}^c, \quad \mathbf{q}_1 = \mathbf{q}_0 + h\mathbf{v}_1, \tag{4.9}$$

where h is the time-step size. We can rewrite this two equations as one

$$\mathbf{M}\mathbf{q}_1 - \mathbf{M}\mathbf{q}_1^0 - h^2\mathbf{F}^c = 0 \tag{4.10}$$

where $\mathbf{q}_1^0$ is the *unconstrained* position vector $\mathbf{q}_1^0 = \mathbf{q}_0 + h\mathbf{M}\mathbf{v}_0 + h^2M^{-1}\mathbf{F}_0^e$; for time steps without collisions $\mathbf{F}^c$ will be computed to be zero and $\mathbf{q}_1 = \mathbf{q}_1^0$.

Discretization of contact constraints and restitution. We choose to separate resolution of contact, i.e., enforcing (4.3), from imposing the restitution condition (4.8). Note: this is not necessarily what we would want to do in the end, but this makes it much simpler to understand the behavior of collision resolution and makes it effectively independent from the choice of physics, as the final time-step velocity is determined by restitution directly. Effectively, the impulse is computed in two steps: (1) an impulse that changes the *pre-collision* velocity so that $\mathbf{q}_1$ is collision-free, and the end position is as close as possible to the unconstrained position (2) adding another impulse changing the post-collision velocity so that it satisfies the restitution equation.

To discretize the constraints (4.3), we specialize to our 2D shape discretization, namely, the union of line segments. Observe that for a pair of segments in a general position, one of the points in the pair of closest points is always a segment endpoint. We conclude that it is

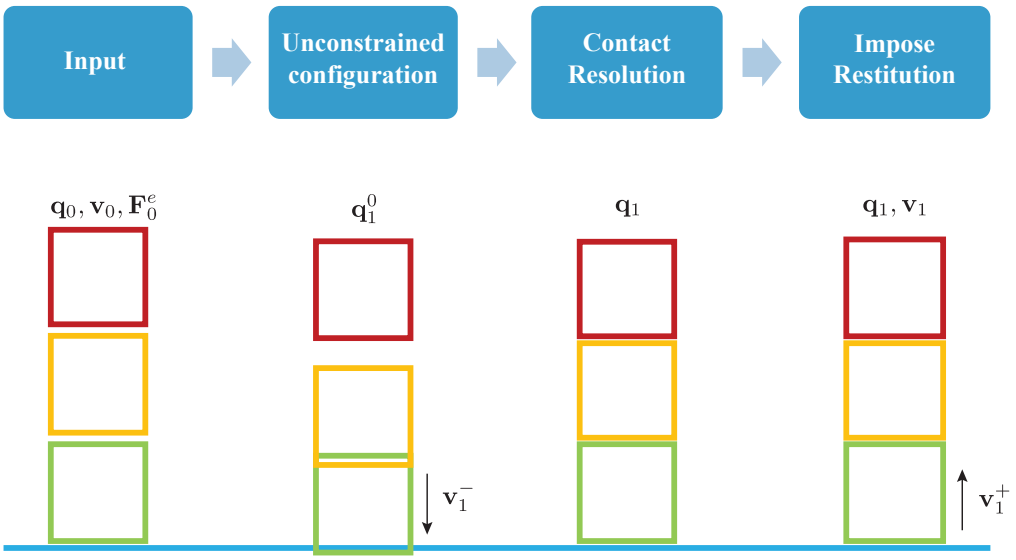


Figure 4.2: Overview of our approach

sufficient to impose a finite number of inequalities

$$d(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k) - \delta_{min} \geq 0 \tag{4.11}$$

for all vertices k and edges $e_{ij} = (\mathbf{x}_i, \mathbf{x}_j)$, where $\mathbf{x}_k$ and e_{ij} belong to distinct objects and $d(\mathbf{x}_i, \mathbf{x}_j, \mathbf{x}_k)$ denotes the minimal distance between the vertex and the edge. This ensures that the infinite set of constraints (4.3) is satisfied. There is no approximation involved in this discretization: the infinite set of constraints is satisfied exactly, and without any reduction in the feasible domain.

At the first step, we are simply minimizing the magnitude of the contact impulse applied. Thanks to (4.10),this is equivalent to minimizing the deviation of the modified configuration $\mathbf{q}_1$ from the unconstrained one $\mathbf{q}_1^0$, with the norm taken with respect to the generalized mass

matrix M, subject to (4.11):

$$\min_{\mathbf{q}_1} ||\mathbf{q}_1 - \mathbf{q}_1^0||_M \quad \text{s.t } d((\mathbf{f}_i(\mathbf{q}_1), \mathbf{f}_j(\mathbf{q}_1), \mathbf{f}_k(\mathbf{q}_1)) - \delta_{min} \geq 0 \tag{4.12}$$

where vertex positions $\mathbf{f}_\ell(\mathbf{q}_1)$, $\ell = i, j, k$, are given by (4.1). The collision forces are obtained as $\mathbf{F}^c = \lambda \nabla d(\cdot)$ with λ the Lagrange multipliers of the constraint. We describe the solver we used for this problem in the next section.

As the second step, we ensure that the post-collision velocities satisfy the restitution equations. We use (4.8) to compute post-collision velocities $\mathbf{v}^+$, using *uncorrected* pre-collision velocities $\mathbf{v}$ as $\mathbf{v}^-$. On the one hand, this ensures that restitution equations for velocity are satisfied exactly. On the other hand, while this, e.g., ensures that the kinetic energy does not change for elastic collisions, the impulse applied to resolve collisions may change the potential energy adding/removing energy to what should be a conservative system. As mentioned above, the two-step separation is not an essential aspect of the approach.

When dealing with multiple collisions in a single time step, we apply impulses sequentially, sorted by time of collision. However, before applying the impulse we need to verify that $\dot{\mathbf{r}}_{rel}^- < 0$, i.e bodies are approaching. For the first impact, this will always be true. However, for following collisions, the previously applied impacts might have altered velocities such that $\dot{\mathbf{r}}_{rel}^- > 0$, i.e bodies are already moving apart. In this last case, we do not apply a new impact. For multiple *concurrent* collisions, our algorithm will arbitrarily choose one to handle first, possibly breaking symmetries.

4.5 Constraint solver

We formulate the problem (4.12) in a more general form, that applies to settings other than rigid body.

We assume that the configuration $\mathbf{x}$ of the vertices is given by $\mathbf{x} = \mathbf{f}(\mathbf{q})$. In our case $\mathbf{q} \in \mathbb{R}^{3M}$ describes the position and orientation of the rigid bodies, and $\mathbf{f}$ is defined by (4.1) . More generally, $\mathbf{f}$ can be understood as a function that maps a low-dimensional configuration $\mathbf{q}$ to a high-dimensional configuration $\mathbf{x}$, e.g., any generalized coordinates in a reduced basis representation. For the purpose of our algorithm, we expect $\mathbf{f}$ to be a differentiable smooth function.

We additionally assume that between two consecutive steps t_0 and t_1, the trajectory of the vertices is linearly interpolated:

$$\mathbf{x}_1(t) = \mathbf{x}_0 + t\Delta\mathbf{x}_1, \quad \text{for } 0 \leq t \leq 1 \tag{4.13}$$

where $\Delta\mathbf{x}_1 = \mathbf{x}_1 - \mathbf{x}_0$. This also defines a trajectory for every segment which is used to detect collisions.

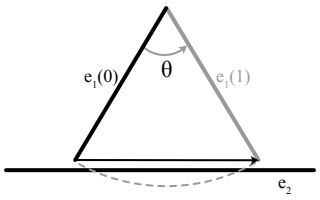


Figure 4.3: We can miss collisions of rotating bodies when we linearize the trajectory.

Note: in the case of rigid bodies, this is an approximation of the trajectory implied by fully rigid motion, so in principle, especially for large steps, this may lead to missed collisions or spurious collisions detected. Using our standard CCD approach, this linear trajectories can always be detected. Angular trajectories induced by rigid body rotation, however, may cause sweeping impacts to be missed (see Figure 4.3). [77] propose a method of CCD for screwing motions of rigid bodies, we leave the integration of their work into ours as a possible future direction.

Problem formulation Given an initial, collision-free configuration $\mathbf{q}_0$ and a unconstrained *candidate* configuration $\mathbf{q}_1^0$, we want to find a collision-free optimal configuration $\mathbf{q}_1$. The general form of the constrained optimization problem we solve is

$$\min_{\mathbf{q}} E(\mathbf{q}) \quad \text{s.t. } d_{ijk}(\mathbf{q}) = d(\mathbf{f}_i(\mathbf{q}), \mathbf{f}_j(\mathbf{q}), \mathbf{f}_k(\mathbf{q})) \geq \delta_{min} \tag{4.14}$$

where $\mathbf{f}_m$ denotes the components of $\mathbf{f}$ corresponding to vertex m, k goes over all vertex indices and (i, j) over all edges. We also drop the index of $\mathbf{q}_1$ to reduce clutter; in the rest of this section $\mathbf{q} = \mathbf{q}_1$.

We observe the following properties of this problem:

- The objective function is smooth and convex;
- The constraints are C^1 away from zero level set of d_{ijk}, (but not C^2); the functions d_{ijk} are convex, although the constraints are not (as these are of the form $d_{ijk} - \delta_{min} \geq 0$).

Remark. one could, relatively easily, build a C^2 or C^∞ function with properties similar to d_{ijk} to ensure continuity of the search directions in the Newton solve.

We note that the total number of constraints is large (quadratic in the mesh size), however, as we discuss below, only few constraints are actually used in the solver at any given time.

We use a barrier-based feasible interior point method. We note that for our problem an initial feasible solution is always available (assuming we do not aim to disentangle an incorrectly specified initial configuration) and phase-I or infeasible solution methods are not essential.

The basic idea of the barrier method is to approximate the constrained problem with an unconstrained problem, which can be solved using Newton's method, by adding extra barrier terms

$$\min_{\mathbf{q}} E(\mathbf{q}) + \frac{1}{t} \sum \phi(d_{ijk}(\mathbf{q})), \tag{4.15}$$

where the parameter $t > 0$ controls the accuracy of the approximation of the solution of the original problem (larger t results in closer approximation), and the summation is over all triples of vertices (i, j, k) such that the first two form an edge. $\phi(x)$ is scalar *barrier* function, that has the following properties: (1) it is at least twice differentiable for all x; (2) it is monotonically decreasing in x, for $x \geq 0$; (3) its limit at zero is infinity.

There are two main features that distinguish our solver from a standard method of this type.

1. We use a barrier function with compact support, unlike standard log-based barrier formulations. This is essential for performance; as the number of constraints is quadratic in the problem size, a standard barrier method would be prohibitively expensive. When evaluating the function value, gradient and Hessian we detect which barrier terms should be included.

2. We use collision detection in the innermost loop of the solver to ensure that we always stay in the feasible domain.

The overall structure of the solver consists of three nested loops: an outer loop increasing the locality and steepness of the barrier, the Newton-step loop determining the function descent direction, and the innermost line search loop, for finding the minimum along the descent direction. We consider it in more detail in the next section.

4.5.1 Solver details

To simplify the notation later on, we rewrite problem (4.15) as

$$\min_{\mathbf{q}} E^b(\mathbf{q}, t) = E(\mathbf{q}) + \frac{1}{t} B(\mathbf{q}) \tag{4.16}$$

where the function B is defined as

$$B(\mathbf{q}) = \sum_{e_{ij} \in S} \sum_{v_k \in V} \phi(d_{ijk}(\mathbf{q})),$$

As mentioned before, problem (4.16) is an approximation of problem (4.14), and its solution approaches the solution of the original problem as t increases. However, for large values of t the problem is very stiff and may require many iterations of a nonlinear solver [20]. To avoid this problem, the standard barrier method solves a sequence of problems (4.16), increasing t on each iteration. Each iteration is solved using Newton's method, and its result is used as a starting point of the next iteration. For a given iteration, i.e fixed t, the precision of the result will be approximately m/t, where m is the number of *active* constraints, i.e., constraints which are nonzero close to the solution.

Outer loop. The outer loop (Algorithm 1) has 4 parameters: initial configuration $\mathbf{q}_0$ user-specified target accuracy ϵ_b, t_0, the initial value of t and μ, the increment of t at each step. The algorithm also uses m, an estimate of the number of active constraints, but in current implementation it is set to $m = 1$, which means that for a larger number of constraints same ϵ_b produces lower accuracy results. The choice of t_0 and μ is discussed in Section 4.5.2.

Newton algorithm. Each internal optimization problem is solved using unconstrained Newton's Method while ensuring we remain in a collision-free state. For the first outer iteration, the initial guess is given by the position and the beginning of the step $\mathbf{q}_0$, which is collision-free by definition. Subsequent iterations start from the solution of the previous problem, which are collision-free by construction: during the line search step, we ensure no collision is generated.

Our implementation of Newton's method is fairly standard (Algorithm 2). The parameters

Algorithm 1 Outer loop

```
1: procedure BarrierMethod(q, t_0, ε_b, μ)
2:     t ← t_0 > 0
3:     q ← q_0                                  ▷ The initial position is collision free
4:     while m/t > ε_b do
5:         q ← arg min_q E^b(q, t)              ▷ Solved using Newton's Method
6:         t ← μt
7:     end while
8:     return q
9: end procedure
```

of the algorithm are $\mathbf{q}_0$ – the initial value, the maximal number of iterations N, the termination accuracy ϵ, explained in Section 4.5.2. We set N to ... to ensure termination in reasonable time in all cases. We note that the solution remains feasible even if the maximal number of iterations is reached, but may be inaccurate.

Line search loop. The innermost iteration is the line search (Algorithm 4), with a starting point $\mathbf{q}$ and a search direction $\Delta\mathbf{q}$, we look for a point $\mathbf{q}' = \mathbf{q} + \alpha\Delta\mathbf{q}$ that satisfies $E^b(\mathbf{q}', t) \leq E^b(\mathbf{q}, t)$. On each line-search step, α is divided by two until this condition is satisfied or the step, $\alpha\mathbf{p}_k$, becomes too small. If no satisfying point is found before that, we stop the current optimization. In this case, the end point is $\mathbf{q}_k$, which can be sup-optimal but is guaranteed to be collision-free.

With this approach, we ensure that every outer iteration of Algorithm 1 ends in a valid, collision-free, state. Therefore, for any choice of ϵ_b, the algorithm will end in a valid simulation state without collisions.

Algorithm 2 Newton algorithm loop

procedure NEWTONSOLVE($\mathbf{q}_0$,N, ϵ)
  $\mathbf{q} \leftarrow \mathbf{q}_0$
  for $i = 0, \ldots, N$ **do**
    $\Delta\mathbf{q} \leftarrow$ NEWTONDESCENTDIRECTION($\mathbf{q}$) ▷ See Algorithm 3
    if $|\nabla_{\mathbf{q}} E^b(\mathbf{q}, t) \cdot \Delta\mathbf{q}| < \epsilon$ **then** ▷ Found local optimum
      return q
    end if

    found, $\alpha \leftarrow$ LINESEARCH($\mathbf{q}, \Delta\mathbf{q}$) ▷ See Algorithm 4
    if not found **then**
      $\Delta\mathbf{q} \leftarrow -\nabla_{\mathbf{q}} E^b(\mathbf{q}, t)$
      if $|\nabla E^b(\mathbf{q}, t) \cdot \Delta\mathbf{q}| < \mathbf{max}\{\epsilon_b, c\frac{m}{t}\}$ **then** ▷ Found local optimum
        return q
      end if
      found, $\alpha \leftarrow$ LINESEARCH($\mathbf{q}, \Delta\mathbf{q}$)
      if not found **: break**
    end if

    $\mathbf{q} \leftarrow \mathbf{q} + \alpha\Delta\mathbf{q}$
  end for
  return q
end procedure

Algorithm 3 Newton Descent Direction

procedure NEWTONDESCENTDIRECTION($\mathbf{q}$)
  $\mathbf{H} \leftarrow \nabla^2_{\mathbf{qq}} E^b(\mathbf{q}, t)$
  $\Delta\mathbf{q} \leftarrow \mathbf{H}^{-1}(-\nabla_{\mathbf{q}} E^b(\mathbf{q}, t))$

  if $\Delta\mathbf{q} \cdot \nabla_{\mathbf{q}} E^b(\mathbf{q}, t) \geq 0$ **then** ▷ Check if $\Delta\mathbf{q}$ is a descent direction
    $\lambda \leftarrow \mathbf{max}\left(\left\{\left(\sum_{j \neq i} |H_{ij}|\right) - H_{ii} \mid 0 \leq i \leq \dim(\mathbf{q})\right\} \cup \{0\}\right)$
    $\mathbf{H}_\lambda \leftarrow \nabla^2 E^b(\mathbf{q}, t) + \lambda \mathbf{I}$ ▷ Make the matrix positive diagonally dominant
    $\Delta\mathbf{q} \leftarrow \mathbf{H}_\lambda^{-1}(-\nabla_{\mathbf{q}} E^b(\mathbf{q}, t))$
  end if

  return $\Delta\mathbf{q}$
end procedure

Algorithm 4 Linesearch Implementation

procedure LINESEARCH($\mathbf{q}$)
  $\alpha \leftarrow 1$
  found$\leftarrow$ **false**
  while $|\nabla_{\mathbf{q}} E^b(\mathbf{q}, t) \cdot \Delta\mathbf{q}| \alpha \geq \mathbf{min}\{c\epsilon_b, 10^{-12}\}$ **do**
    $\mathbf{q}_i \leftarrow \mathbf{q} + \alpha\Delta\mathbf{q}$
    if $E^b(\mathbf{q}_i, t) < E^b(\mathbf{q}, t)$ **and** COLLISIONFREE($\mathbf{q}, \mathbf{q}_i$) **then**
      found $\leftarrow$ **true**
      break
    end if
    $\alpha \leftarrow \frac{\alpha}{2}$
  end while
  return (found, α)
end procedure

4.5.2 Parameter choices

Choice of the barrier function. We have experimented with two options for the barrier function. As mentioned above, it is essential to use a compactly supported function. The trade-off is between the performance (smaller support leads to fewer simultaneously active constraints), and stiffness and numerical stability of the unconstrained system. The two main functions we have considered are

- A rational barrier $\phi_\epsilon^{spl}(x) = 1 - 1/g(x/\epsilon)$, with $g(y) = y^3 - 3y^2 + 3y$;
- The function $\phi_\epsilon^{log}(x) = \phi_1^{log}(x/\epsilon)$ obtained by limiting the support of the logarithmic barrier.

$$\phi_1^{log}(y) = \begin{cases} \infty, & y \le 0 \\ -\log(y)(2y^3 - 3y^2 + 1), & 0 < y < 1, \\ 0, & y \ge 1 \end{cases}$$

The first function has the advantage that it does not require transcendental functions. However, it behaves asymptotically as $1/y$ as $y \to 0$, and therefore behaves as an inverse barrier function. Inverse barriers result in slower convergence of the solver: we have observed roughly $\sqrt{1/t}$ decrease in the solution error, vs $1/t$ decrease for the logarithmic solver.

Initial value t_0 and increment μ of t. The initial value of t, following, e.g., [20] can be obtained by minimizing $\|\nabla E(\mathbf{q}_0) + t\nabla B(\mathbf{q}_0)\|^2$ with respect to t, unless $B(\mathbf{q}_0) = 0$ in the initial configuration, in which case we simply set it to 1. Larger values of μ result in the increase in Newton iterations, but fewer outer iterations for a given accuracy. From our parameter sweeps we observe that $\mu = 100$ is a reasonable choice

Newton solver accuracy ϵ. We set this parameter based on the parameters of the outer loop as $\epsilon = \mathbf{max}\{\epsilon_b, c\frac{m}{t}\}$, where c is a heuristic constant. This is based on the following

considerations: at the early outer iterations, the estimated accuracy of the solution m/t is low, and it does not make sense to solve the unconstrained minimization problem too precisely, as long as the solution remains feasible. On the other hand, if we set the accuracy too low, this may result in extra iterations at following steps. The constant $c \leq 1$ tries to achieve the balance between these two aspects. At the same time, once t is large enough, the value of cm/t may be lower than the target accuracy ϵ_b, and there is no need to set the Newton solver accuracy higher than this.

4.5.3 Relation to complimentarity problems

The connection between the penalty formulation and the complimentarity problems is standard, and elucidates the relation to common collision techniques. The solution to the unconstrained optimization problem with fixed parameter t satisfies

$$\nabla E + \frac{1}{t} \sum B'(d_{ijk}(\mathbf{q})) \nabla d_{ijk}(\mathbf{q}) = 0 \tag{4.17}$$

Compare to the KKT system:

$$\nabla E - \frac{1}{t} \sum \lambda_{ijk} \nabla d_{ijk}(\mathbf{q}) = 0, \tag{4.18}$$

$$d_{ijk}(\mathbf{q}) \geq 0; \lambda_{ijk} \geq 0 \tag{4.19}$$

$$\lambda_{ijk} d_{ijk}(\mathbf{q}) = 0 \tag{4.20}$$

Setting $\lambda_{ijk} = -B'(d_{ijk})/t$ at the optimum, we observe that (4.17) becomes identical to the first KKT equation. We also observe that with the logarithmic barrier function, λ_{ijk} is bounded (and approaches $1/(td_{ijk})$ as t grows, and as a consequence, distances at the optimum decrease. It is clearly positive, and as the product with d_{ijk} approaches $1/t$, also satisfies the complimentarity condition in the limit. We also note that the barrier term in

the gradient is a sum of forces of the form $\lambda_{ijk}\nabla d_{ijk}$ for each active constraint (i.e., with nonzero barrier). As t increases, the minimum point approaches contact, and accordingly the gradient of the distance approaches the normal to the surface, if the surface is smooth, i.e. the contact forces approach the commonly used form λn.

Finally, common linear complimentarity conditions are a linearized form of the conditions above, with the distance function near zero value replaced by the linear part in its expansion $n(\mathbf{q}_s) \cdot (\mathbf{q} - vq_s)$.

4.6 Results

4.6.1 Effect of parameters

Heuristic constant c. The constant $c \leq 1$ is used to balance the performance and accuracy of the Newton iterations. A large value of c indicates we are solving Newton to the same accuracy as the outer loop, which might be low in the first barrier iterations; this may help with performance, decreasing the number of iterations on the first loop. On the other hand, if the first iterations are too inaccurate, the Newton solves at subsequent iterations might take more time. However, from our experiments, using $c = 1$ results in best performance (indicated by the number of functional, hessian, and CCD evaluations), i.e., there is no significant effect on the number of Newton iterations on subsequent steps of the outer loop. The plot in Figure 4.4 shows the results for the Billiards scene, using $h = 0.01$, $C = 1$, $e_b = 10^{-6}$, $m = 1$, $t_0 = 100$, $\mu = 100$, and $c \in \{10^{-i}, i = -4 \dots 0\}$.

Initial value t_0 and increment μ. To evaluate the effect of parameters t_0 and μ, we ran a sweep of the parameters: $t_0, \mu = 10^i, i = 0 \dots 6$, adding the value $\mu = 2$ for each two. The image below shows the results for the Billiards scene, using $h = 0.01, C = 1$. We expected that large values of μ would require more Newton iterations (larger number of function

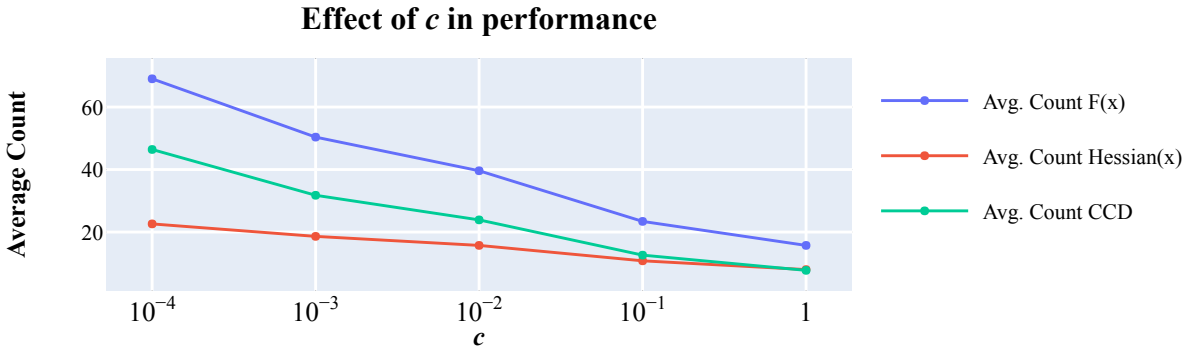


Figure 4.4: The plot shows the effect of the constant c in the performance of our optimization, measured in terms of number of evaluations of the functional $f(x)$, hessian $\nabla^2(x)$ and the CCD subroutine.

evaluations) but we were unable to find evidence for this in our set of examples in frictionless setting. Our tests consistently showed that using a large value of t_0 and μ converged well. One can explain this that our initialization, $\mathbf{q}_0$, is typically near the optimal solution $\mathbf{q}_1$. In other words, by initializing the solution with the initial position of the time-step, we make it possible for the Newton method to solve the equations in few iterations, despite the system being very stiff for large t.

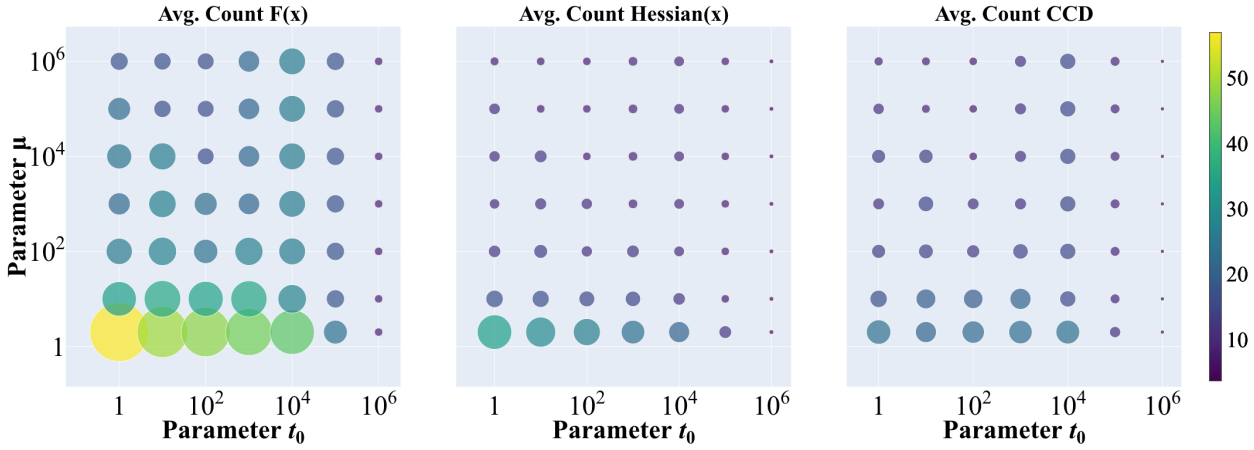


Figure 4.5: These plots show the effect of t_0 and μ on the number of evaluations of the functional $(F(x))$, its Hessian $(Hessian(x))$, and the CCD subroutine. We use the data from the Billiards scene, using $h = 0.01, C = 1$.

User-specified accuracy ϵ_b. As mentioned in section 4.5.1, the parameter ϵ_b controls the accuracy of the solution. To measure its effect, we test our algorithm with different values for ϵ_b and measure the minimum distance over all time-steps. The plot on Figure 4.6 shows this data for the Newton's Cradle scene, using $h = 0.01, C = 1$. As expected, the accuracy of the solution improves linearly when decreasing ϵ_b.

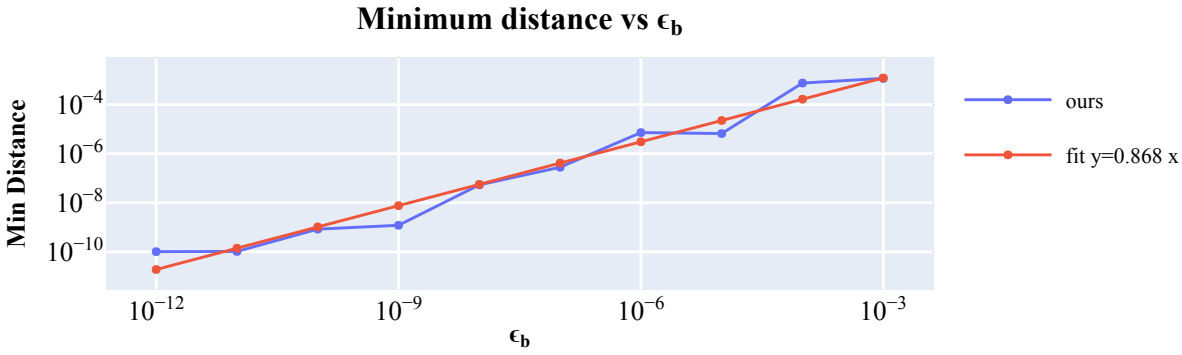


Figure 4.6: Plot shows the linear relationship between the minimum distance (as a proxy for constraint enforcement accuracy, for timesteps with collisions) and the value ϵ_b: the accuracy of our result is larger for smaller values ϵ_b. Data based on the Newton's cradle simulation using $h = 0.01, C = 1$.

In Figure 4.7, we show the effect ϵ_b on the final resting configuration of a pyramid of blocks. We can observe small differences in the rest configuration: for a large $\epsilon_b = 10^{-2}$, the green square lies further from the ground; for $\epsilon_b = 10^{-4}$, the distance between the green square and the ground is smaller than what we can visualize. The bottom right image on Figure 4.7 shows the solutions superimposed; red corresponds to $\epsilon_b = 10^{-2}$, green to $\epsilon_b = 10^{-4}$, and blue to $\epsilon_b = 10^{-6}$.

Independently of ϵ_b, the final configuration is collision-free and without jittering.

Non-straight collision-free trajectories. During the line-search step, we check for collisions between the last Newton step configuration, $\mathbf{q}$, and the line-search candidate $\mathbf{q}_i$. This means the trajectory from $\mathbf{q}_0$ to $\mathbf{q}_1$ will be only *piece-wise* linear. In other words, the linearized trajectory from $\mathbf{q}_0$ to $\mathbf{q}_1$ might have collisions but the steps taken to get from $\mathbf{q}_0$ to

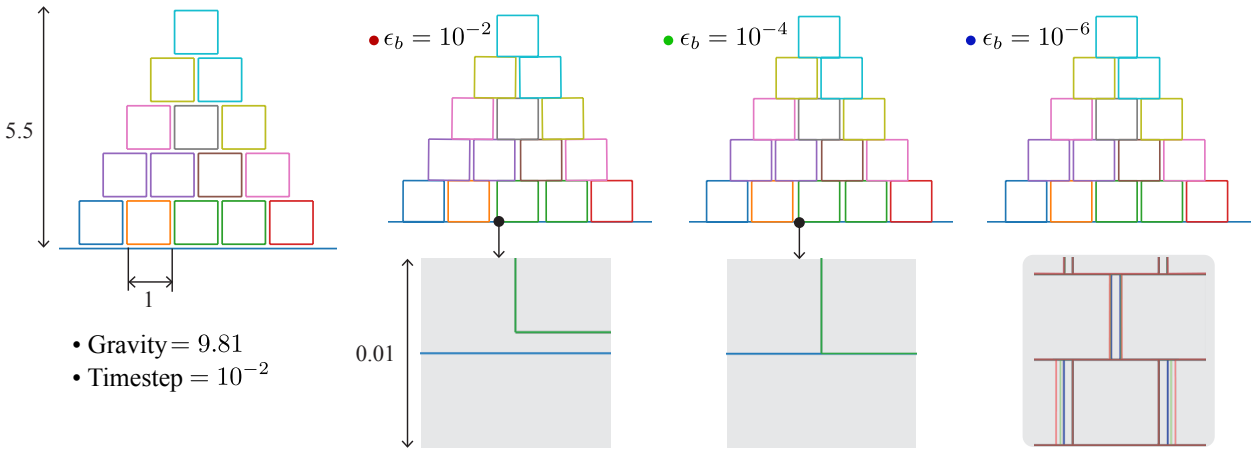


Figure 4.7: Effect of ϵ_b parameter on the final resting distance of a pyramid of blocks.

$\mathbf{q}_1$ are all free of collisions. This may lead to unintuitive time-steps (see Figure 4.8), but we view allowing such piece-wise linear collision free trajectories as an affect of not visualizing the inner optimization iterations. Note that this does not mean the solver can take any arbitrary path from $\mathbf{q}_0$ to $\mathbf{q}_1$, because each Newton step follows the gradient direction, the path will be approximately an integral line of the gradient field.

To illustrate this effect, we tested a box falling onto a corner of thin edge. Figure 4.8 shows the simulation steps (colored by time) for different velocities and positions. For large velocity and a position near the boundary, our optimization finds a piece-wise collision-free trajectory bending around the thin edge. For slightly smaller velocities, or with a position farther from the edge corer, the optimization results in a collision free trajectory from $\mathbf{q}_0$ to $\mathbf{q}_1$.

4.6.2 Evaluation and comparisons with Box2D and STIV-NCP

To validate our approach, we performed a comprehensive evaluation on a set of 18 scenes (Figure 4.20). We tested our method and two previous approaches: Box2D, and our own implementation of a version of the STIV-NCP solver presented in [59].

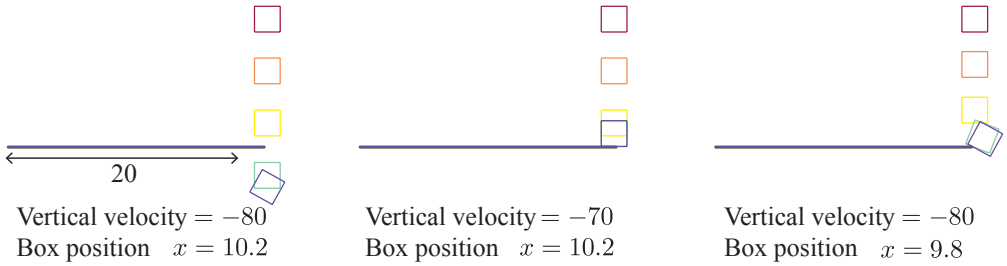


Figure 4.8: Effect of piece-wise collision free trajectories. Images show all frames of the simulation, colored by time. Left: on the third time step our optimization finds a piece-wise collision free trajectory around the line. Center: decreasing the box's velocity results on the box been stopped by the segment. Right: alternatively, keeping the velocity the same as the first scene, and moving the box slightly to the left has a similar effect.

For all examples shown in this chapter, we configure Box2D to use use 10K maximum velocity, 10K maximum position iterations, and no friction. For our algorithm, all examples use $e_b = 10^{-6}$, $m = 1$, $c = 0.1$, $t_0 = 100$, $\mu = 100$, unless otherwise mentioned.

STIV-NCP implementation details In our implementation, we have changed the space-time volume to a more geometrically accurate discretization scheme. For a collision between edges ij and kl we define two volumes, one for each edge. For each edge ij (resp. kl) the space-time volume V_{ij} (resp. V_{kl}) is defined as

$$V_{ij}(\mathbf{e}, \tau_I, \alpha_{ij}) = (1 - \tau_I)\sqrt{\epsilon^2 \|\mathbf{e}_{ij}(\tau_I)\|^2 + (\Delta\mathbf{x}_{ij} \cdot \mathbf{e}_{ij}(\tau_I)^\perp)^2} \tag{4.21}$$

where $\tau_I \in [0, 1]$ is the time of impact, $\alpha_{ij} \in [0, 1]$ is the position along the edge where the collision happened, $\Delta\mathbf{x}_{ij} = \Delta\mathbf{x}_i + \alpha_{ij}(\Delta\mathbf{x}_j - \Delta\mathbf{x}_i)$ corresponds to the displacement of that point, $\mathbf{e}_{ij}$ is the edge length, and ϵ is a constant set to 10^{-10}. This definition is a discretization of the general space-time volume definition.

We implement a simplified version of the minimum distance constraint, my modifying the time of impact, based on velocities of vertices so that the reported collision corresponds to the vertex being at a distance at least 10^{-6} distance units from the edge. The solver

implementation is detailed in Algorithm 5. To solve the LCP problem we use our own Gauss-Seidel implementation based on [26]. In the rare cases where the algorithm is not converging (number of iterations exceeds 20), we use the edge perpendicular as gradient direction, as it was done in the implementation of [59]. We note that [59] uses the minimial-map Newton method, rather than a Gauss-Seidel LCP solver. We do not expect using this solver to significantly improve the range of cases which the NCP method can handle, but it may significantly speed up performance and allow the method to complete more simulations.

Robustness For each scene, we tested our simulation and prior approaches with time-steps h set to 0.1, 0.01, and 0.001 and the coefficient of restitution $C = 0$ or 1. Figure 4.19 shows the results of these tests grouped by scene. Each simulation was given 10hrs to complete a number of simulation steps (column 2, `target #iterations`). We record the total number of iterations which finished in that time, and the number of iterations that had interpenetrations of objects at the beginning (column 4, using format `#steps with interpenetration/#steps`), i.e., these were not resolved by the previous step.

On Figure 4.19, green highlights correspond to simulations that finished the total number of iterations without generating interpenetrations; red to simulations that finished but had interpenetrations; gray to simulations that did not finished; and yellow to simulations that finished without interpenetrations but other constraints were violated.

We can observe that while STIV-NCP did not complete some of the scenes on time (highlighted in grey), it was able to solve collisions on the time steps it completed in most cases. The only exception is the Compactor scene, where solving collisions for the large box is complicated, as any motion not perfectly aligned with the x-axis generates collisions. We also observe that for several scenes only few steps could be performed.

On the other hand, Box2D was able to finish all simulations in time but it fails to solve collisions on 37 scenes (34%, highlighted in red). Moreover, in the cogs scenes Box2D is unable

to keep the position of the cogs constant (highlighted in yellow). Based on the data we have collected, Box2D seems to work better (less number of steps with unresolved interpenetration) for smaller time-steps. In contrast, our method solves collisions robustly and all scenes, independent of time-step or coefficient of restitution used.

Accuracy of the solution. To compare the accuracy of constraints for different methods, we plot the minimum distance between objects over all time-steps (Figure 4.9); for steps with collisions, minimum distance is a measure of how close the solution is to the true solution.

We should note that by default Box2D uses a small gap of 0.005 between bodies for numerical stability. For all scenes in the presence of collisions, our solver consistently results in a smaller minimum distance than Box2D (i.e., the end configuration is closer to satisfying the constraints exactly). Compared to STIV-NCP, we are more reliable in obtaining minimum distances: in some scenes, e.g. Pyramid, STIV-NCP shows a smaller min distance but in others it overshoots by a considerable amount, e.g Billiards. This happens because the termination condition for STIV-NCP does not include any convergence criteria, it only requires no-interpenetration.

Energy conservation. To evaluate how well our system conserves energy, we plot the total energy of the system vs. time. Figure 4.10 shows the energy for the bouncing diamond scene with the coefficient of restitution $C = 1$, and time step $h = 0.01$. For this configuration, the diamond bounces five times on the floor, while moving to the right. The total energy of the system decreases during free fall as a result of the time-integration used (Semi-Implicit Euler), which suffers significant numerical damping for constant acceleration motions. The time steps with collisions can be easily identified in the plot by the increase in energy produced in Box2D and STIV-NCP. The increase in energy is a consequence of having a larger post-collision separation between the diamond and the ground: while the kinetic energy is preserved before

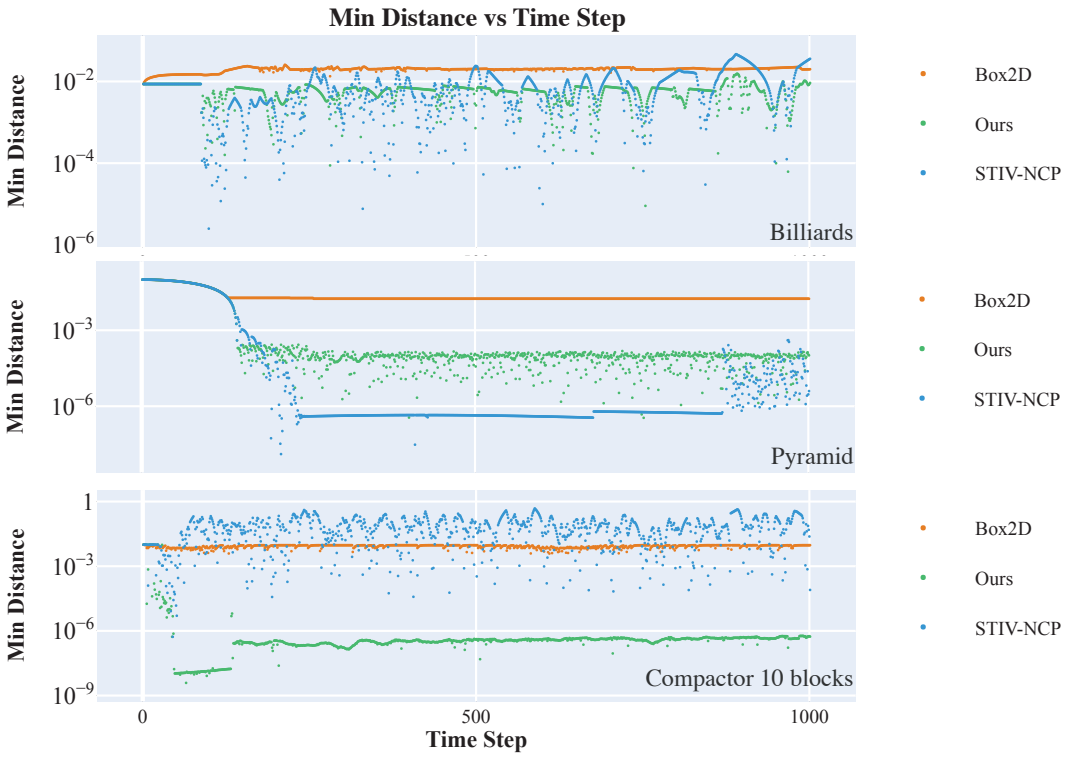


Figure 4.9: Minimum distance at each simulation step for different scenes: Billiards, $C = 1$, $h = 0.001$; Pyramid $C = 0$, $h = 0.001$; Compactor $C = 1$, $h = 0.001$

and after impact (restitution law), the potential energy is increased. Our method is able to preserve potential energy better because it can robustly handle smaller minimum distances.

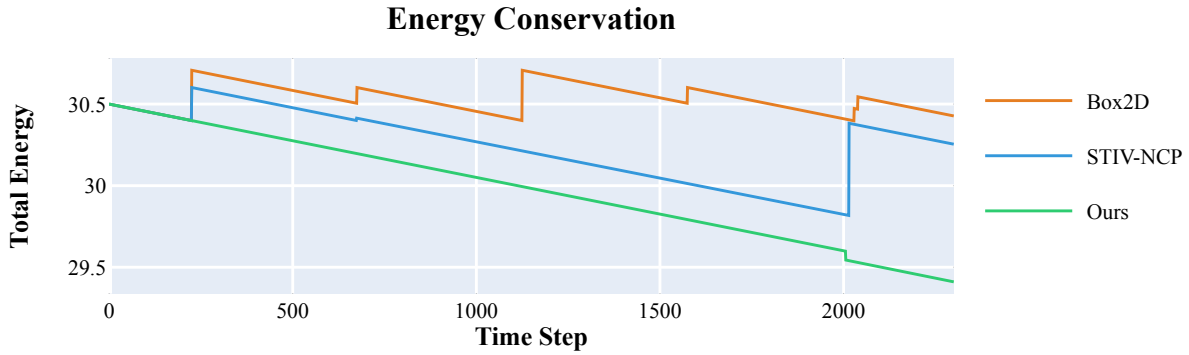


Figure 4.10: We plot the total energy of the system over each simulation step. Compare to Box2d and STIV-NCP, our method is able to better preserve the energy, without injecting more into the system.

Stress test. We place a variable number of small blocks into a "compactor" where a body with large mass and velocity pushes them into a fixed wall. Figure 4.11 illustrates the results for our method and Box2D for a varying number of small blocks. Box2D is unable to handle the high momentum impact and results in many blocks passing through the fixed boundaries. In contrast, we are able to robustly handle the compression and tightly pack the blocks into several columns.

Interlocked chains. Interlocked geometry exhibits an interesting behavior of jamming in which all bodies have reached a stable configuration relying on a single fixed body. Physically, jamming may occur due to friction, so called *frictional jamming*. In our friction-less simulation we observe another form of jamming, geometric jamming, where the geometry and collisions produce a stable configuration under external forces (e.g., gravity). Such cases of jamming are common in 3D, for example fro the interlocking geometry of chain mail.

Our method is able to handle competing forces robustly. We designed a 2D interlocking

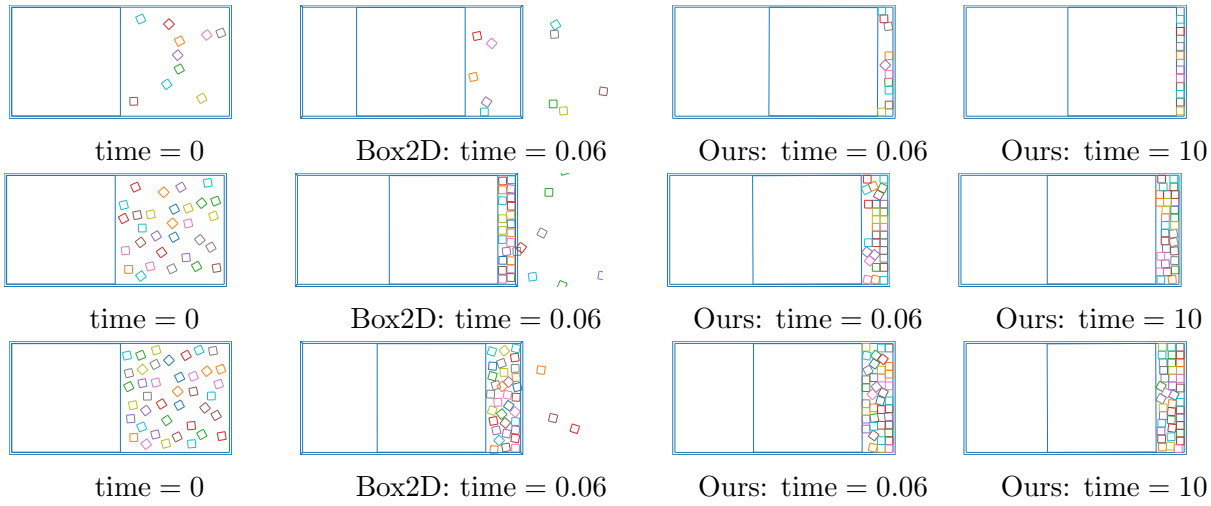


Figure 4.11: A "compactor" with a variable number of blocks using both our method and Box2D. All simulations use $h = 10^{-2}$ and a $C = 1$. The large box is given a mass of 1000 and $\mathbf{v}_x = 100$.

chain of links (Figure 4.12). Moving under gravity, it reaches a state of geometric locking. In order to maintain the static configuration, collision handling must not increase energy, which often has to be achieved by adding artificial damping. Energy injected into the system as a result of collision can cause wave motions down the length of the chain.

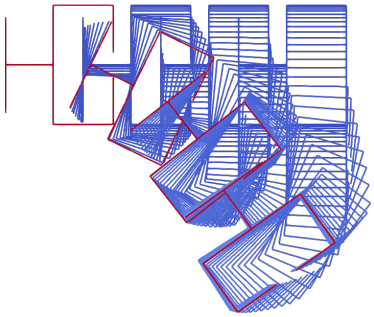


Figure 4.12: Four 2D links are chained together and allowed to relax under gravity. The first link in the chain is fixed in its position and orientation. We visualize all time steps in one image using a color map for the time. Note later times, in red, all have the same position and orientation.

We compare the ability of Box2D and our method to handle such cases under varying time-steps (Figure 4.13). To robustly resolve dynamic scenes, collision handling must be

agnostic to the time step and consequently displacement size. Many collision response methods rely on a small time step to limit the length of displacements. However, it is difficult to choose such step automatically: with no prior bound on velocities, the displacements can be arbitrarily large for any given time step. Our method is able to robustly handle collision for a large range of step sizes.

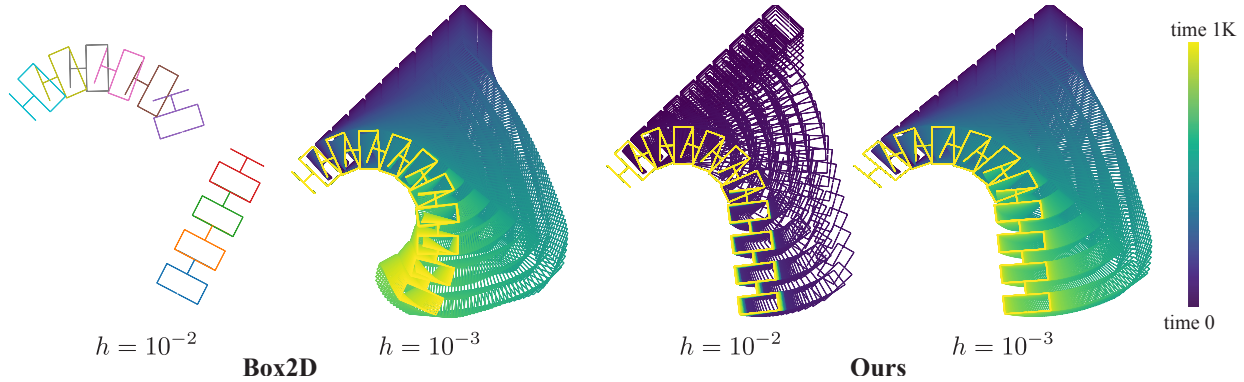


Figure 4.13: Our method works for a broad range of time steps, compared to Box2D. In this figure, positions of an object at several time steps are shown on top of each other.

Cogs. We test another type of geometric jamming using a loop of three cogs (Figure 4.14). In a loop with an odd number of cogs, the alternating clockwise/counter-clockwise rotations prevent the cogs from turning. In our example, the red cog is given an initial angular velocity of -100 radians per second and spins clockwise. First, the red cog impacts the orange one causing it to rotate counter-clockwise. Then, the orange cog collides with the grey cog causing it to rotate clockwise. However, the red cog has already forced the grey cog to rotate counter-clockwise. These opposite motions cancel out and the cogs become jammed. While Box2D does not generate any inter-penetrations in this scene, it violates the position constraints on the cogs. A similar situation occurs when using a line of three cogs (Figure 4.15), where, for $h = 10^{-2}$, Box2D solves collisions but violates positional constraints of the cogs. In contrast, in both scenes, we are able to solve collisions successfully while maintaining the cogs in their original position.

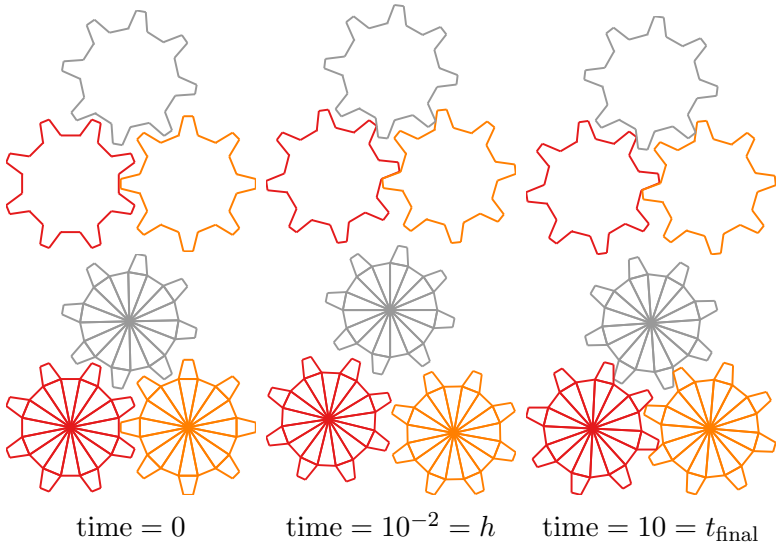


Figure 4.14: This configuration of three cogs causes geometric jamming. The top row shows our results for the first, second, and last time step. The bottom row shows the results of Box2D for the same time steps. The collisions cause Box2D to violate position constraints on the cogs.

Stacking lines. To evaluate our method's ability to handle thin features, we consider a collision of a stack of thin boxes (thickness of 10^{-2}) with a fast-moving box. Although our method is capable of handling single edges, we choose to use thin boxes to allow for comparison with Box2D. Figure 4.16 shows the results of our simulation as well as the STIV-NCP method. For this scene, the STIV-NCP simulation with normal-direction fallback did not finished in time. While the simulation without fallback does finish, it fails to solve collisions.

Newton's cradle. Newton's Cradle is a toy that demonstrates conservation of momentum and energy. The real-world Newton's cradle is composed of a set of suspended balls arranged in a row, and in contact in the rest state. Deflecting the first ball in the chain and letting it hit the second one, results in deflection of the last ball, as a result of the momentum transfer along the chain. The last ball, in turn, hits the penultimate one, causing a new deflection of the first. The process repeats periodically. In our simulation, we show that the momentum

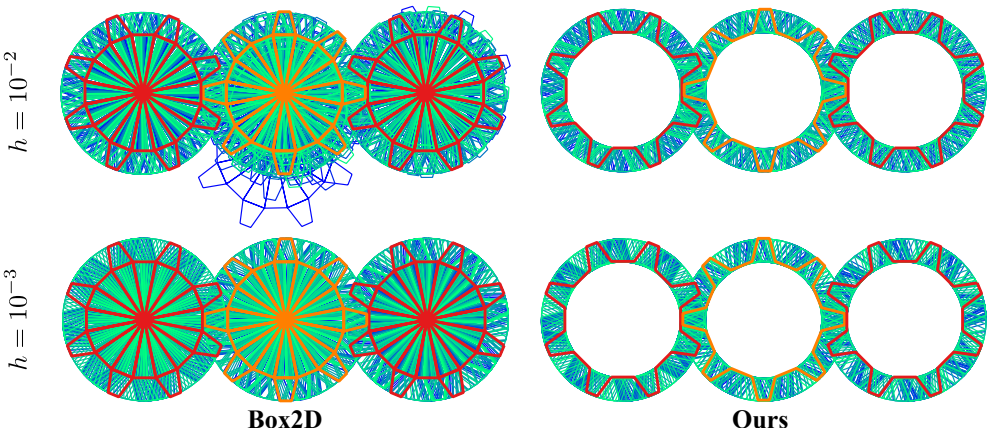


Figure 4.15: We show a line of three cogs, including their initial configuration (red-orange) and all time-step (green-blue). The leftmost cog is initialized with an angular velocity of -100 radians per second, and the rotation quickly propagates to each cog in the line. With $h = 10^{-2}$ Box2D violates positional constraints.

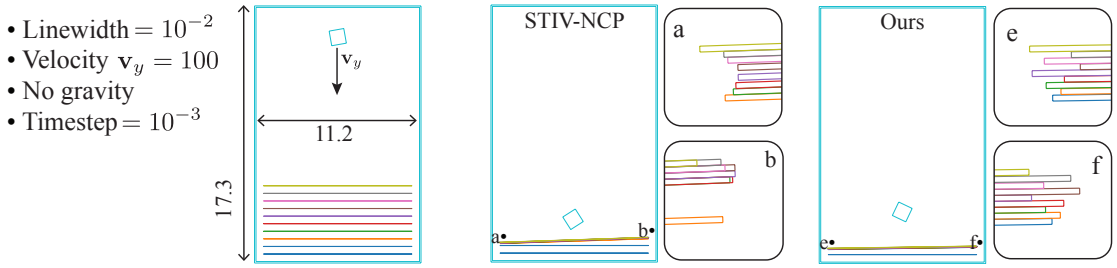


Figure 4.16: A stack of several thin boxes colliding with a fast-moving box. The STIV-NCP methods are unable to resolve the collisions, while our method is able to robustly handle the compressed stack.

transfer works as expected (see Figure 4.17).

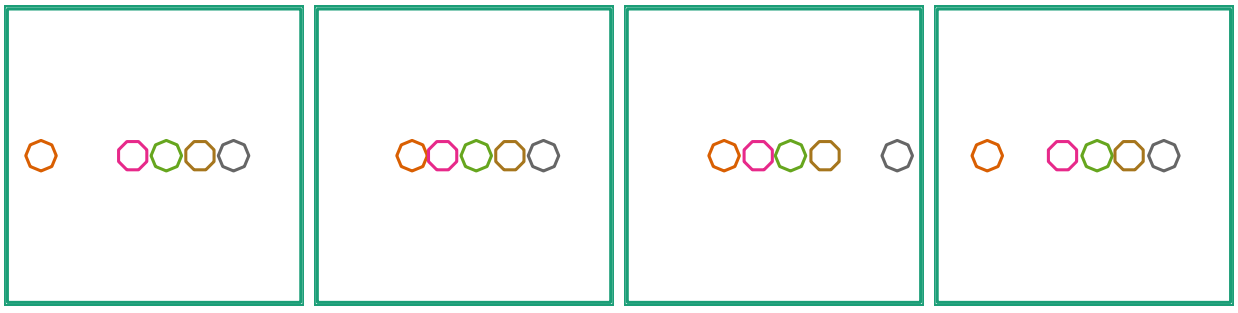


Figure 4.17: Here we show our conservation of energy through a perfectly elastic collision.

Shape parameter sweep An important application of robust collision handling is the ability to sweep shape parameters (e.g., to obtain a desired behavior of the overall system) without the need to tweak simulation parameters. To test our method, we design a parametric chain (Figure 4.18), where we can define the width of the links, S_x, and the height of the neck, D. Our simulation is able to handle these varying designs without the need to tweak simulation parameters. We also shows our ability to handle non-volumetric models, as each segment of a link is represented as a single edge.

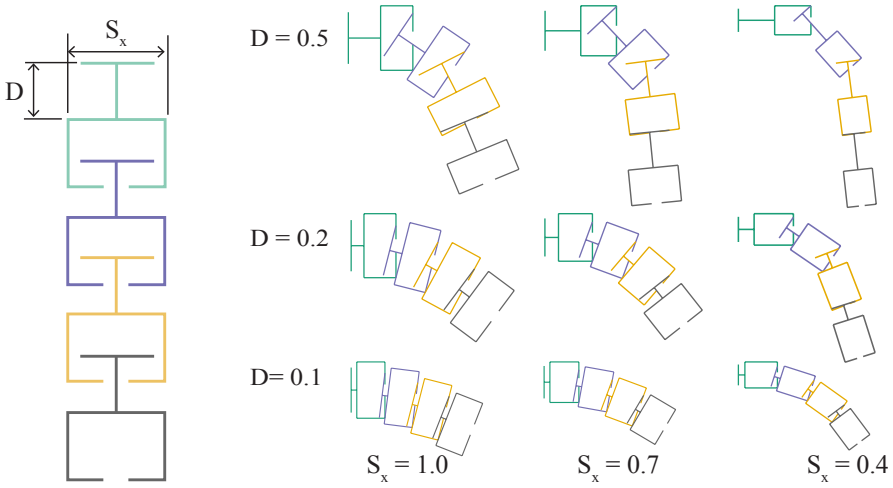


Figure 4.18: Sweep of the design parameters of a chain.

4.7 Limitations and concluding remarks

Our algorithm guarantees that on every time step it produces an interpenetration-free configuration, but the trajectory from the configuration at the previous time step many not be linear; an object can "go around" a thin obstacle if the collision is close to an endpoint, resulting in somewhat unintuitive behavior for large time steps. This can be prevented by changing the collision check condition to verify the linearized trajectory from $\mathbf{q}_0$ to $\mathbf{q}_1$ is collision-free. However, this may decrease the ability of the algorithm to make large time

steps. Empirically, our approach rarely if ever, creates trajectories that suffer from this problem.

More importantly, our algorithm does not currently support *friction*. Modeling friction is necessary in most applications Incorporating friction forces and constraints into our optimization framework will be our next step.

Algorithm 5 NCP Solver

procedure NCP SOLVE(x)
  $A \leftarrow M$
  $b \leftarrow M\mathbf{q_1^0}$
  $x_i \leftarrow A^{-1}b$
  $V_i, \nabla V_i \leftarrow$ GETCOLLISIONVOLUMEGRADIENT(x_i)
  for $i = 0$ to $i <$ max_iter **do**
    if $V_i[e] \geq 0 \, \forall e$ **: break**
    $\lambda_i, \delta_i \leftarrow$ LCPSOLVER$(x_i, V_i, \nabla V_i)$
    $\alpha \leftarrow 1$
    while $||\alpha\delta_i||^2 > 10^{-10}$ **do**
      $V_{i+1} \leftarrow$ GETCOLLISIONVOLUME$(x_i + \alpha\delta_i)$
      if $\sum V_{i+1} > \sum V_i$ **: break**
      $\alpha \leftarrow \alpha/2$
    end while
    $x_i \leftarrow x_i + \alpha\delta_i$
    if $i > 20$ **then** ▷ fallback when algorithm is not converging
      $V_i, \nabla V_i \leftarrow$ GETCOLLISIONNORMAL(x_i)
    else
      $V_i, \nabla V_i \leftarrow$ GETCOLLISIONVOLUMEGRADIENT(x_i)
    end if
  end for
end procedure

Scene	timestep	target #iterations	Box2D		STIV-NCP		Ours	
			cor=0	cor=1	cor=0	cor=1	cor=0	cor=1
Axle	0.1	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
Billiard	0.1	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
Bouncing Diamond	0.1	2300	0/2300	0/2300	0/2300	0/2300	0/2300	0/2300
	0.01	2300	0/2300	0/2300	0/2300	0/2300	0/2300	0/2300
	0.001	2300	0/2300	0/2300	0/2300	0/2300	0/2300	0/2300
Chain	0.1	5000	4994/5000	127/5000	0/1	0/1	0/5000	0/5000
	0.01	5000	46/5000	428/5000	0/54	0/57	0/5000	0/5000
	0.001	5000	0/5000	0/5000	0/437	0/2676	0/5000	0/5000
Chain Net	0.1	1000	137/1000	149/1000	0/3	0/8	0/1000	0/1000
	0.01	1000	15/1000	14/1000	0/110	0/103	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/570	0/859	0/1000	0/1000
Cog Large	0.1	1000	0/1000*	0/1000*	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000*	0/1000*	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
Cog Line	0.1	1000	0/1000*	0/1000*	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000*	0/1000*	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
Cog Loop	0.1	1000	0/1000*	0/1000*	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000*	0/1000*	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000*	0/1000	0/1000	0/1000	0/1000
Compactor 10 boxes	0.1	1000	9/1000	1/1000	1/1000	1/1000	0/1000	0/1000
	0.01	1000	2/1000	4/1000	1/1000	1/1000	0/1000	0/1000
	0.001	1000	15/1000	5/1000	1/1000	1/1000	0/1000	0/1000
Compactor 30 boxes	0.1	1000	13/1000	15/1000	0/1	0/1	0/1000	0/1000
	0.01	1000	64/1000	15/1000	0/0	0/0	0/1000	0/1000
	0.001	1000	37/1000	26//1000	0/26	0/11	0/1000	0/1000
Compactor 60 boxes	0.1	1000	35/1000	14/1000	0/0	0/0	0/1000	0/1000
	0.01	1000	26/1000	43/1000	0/0	0/0	0/1000	0/1000
	0.001	1000	19/1000	44/1000	0/4	0/3	0/1000	0/1000
filling-box	0.1	1000	29/1000	148/1000	0/57	0/123	0/1000	0/1000
	0.01	1000	8/1000	0/1000	0/453	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
Interlocking Saws 10 Teeth	0.1	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
Interlocking Saws 100 Teeth	0.1	1000	0/1000	0/1000	0/6	0/6	0/1000	0/1000
	0.01	1000	0/1000	0/1000	0/101	0/64	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/654	0/656	0/1000	0/1000
Line Stack	0.1	1000	13/1000	15/1000	0/1	0/1	0/1000	0/1000
	0.01	1000	419/1000	17/1000	0/9	0/9	0/1000	0/1000
	0.001	1000	136/1000	3/1000	0/175	0/136	0/1000	0/1000
Newton's cradle	0.1	1000	4/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
Pyramid	0.1	1000	20/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
Saw	0.1	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.01	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000
	0.001	1000	0/1000	0/1000	0/1000	0/1000	0/1000	0/1000

Figure 4.19: We tested the different methods on 18 scenes, with 6 configurations (Figure 4.20). On the table, green highlights correspond to simulations that finished the total number of iterations without generating interpenetrations; red to simulations that finished but had interpenetrations; gray to simulations that did not finished; and yellow to simulations that finished without interpenetrations but other constraints were violated.

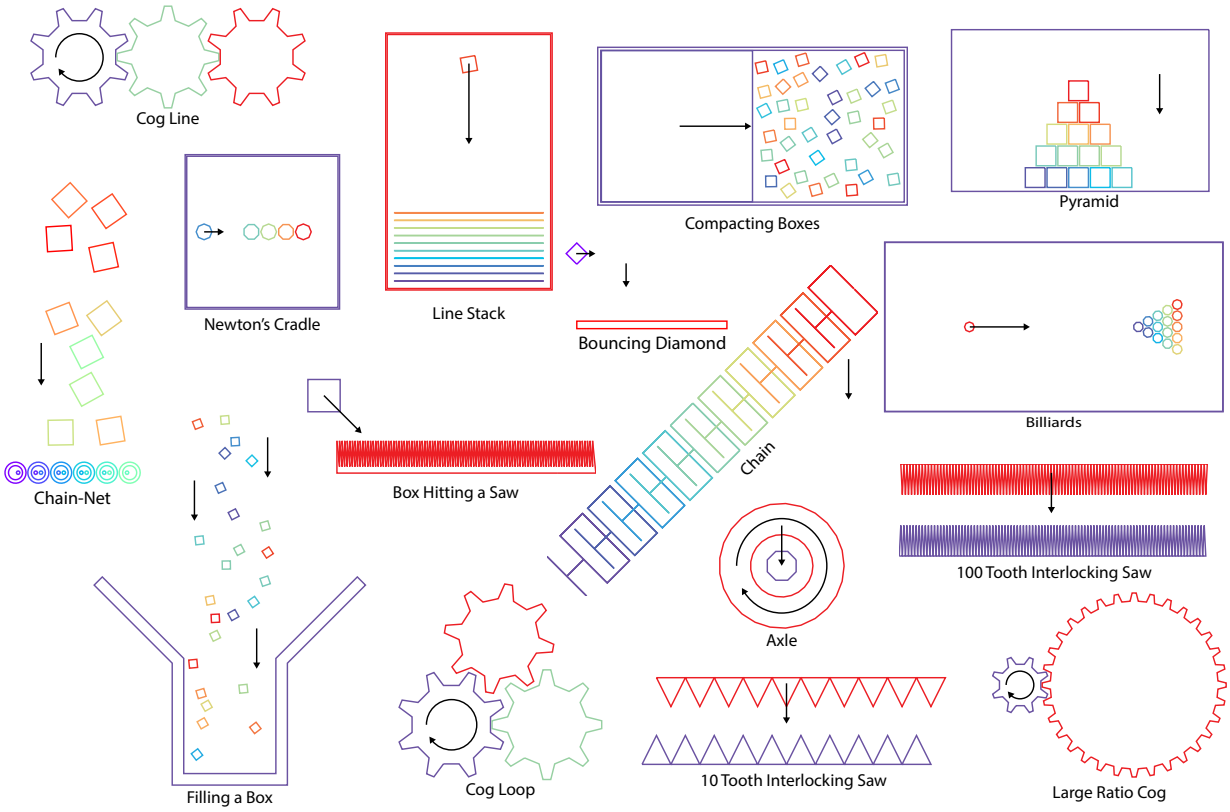


Figure 4.20: We evaluate our method on scenes with varying geometry, scale, complexity, velocities, and masses.

Chapter 5

Conclusion

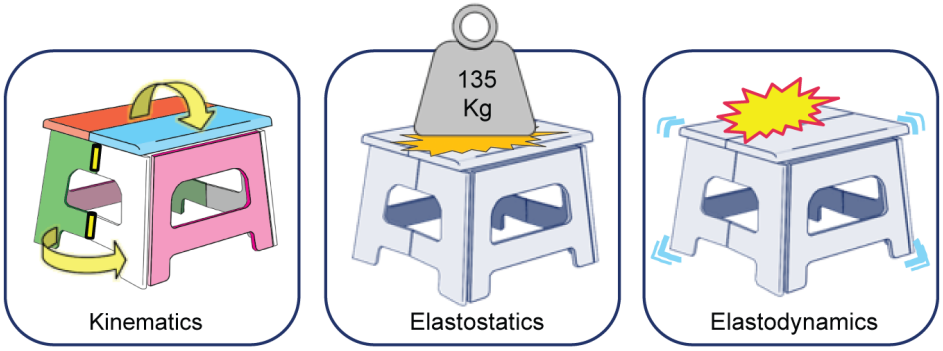


Figure 5.1: This thesis study three aspects of an objects functionality: kinematics, elastostaics, and elastodynamics.

This thesis has presented three works related to different aspects of computational design. To place these in a common context, we consider a simple example of a folding stool design (Figure 5.1). Chapter 2 helps us design the *kinematics* behaviour of a mechanisms, and automatically creates the necessary geometry. In the context of our example, (Figure 5.2), our work would help explore the design space of opening and closing motions, and generate the geometry for the hinges. In the future, we would like to combine our method with stability and strength analysis to ensure, for example, that the opened stool configuration locks under vertical loads.

While Chapter 2 provides a convenient framework for joint design, by itself cannot ensure the stool is strong enough to sustain the weight of a person.

We study this problem in Chapter 3, and present an efficient algorithm for optimizing shell reinforcements under static loading (*elastostatics*). We separate the optimization into two stages, layout and shape optimization. The optimal layout is agnostic to material and scale, so it can be re-utilized for different manufacturing pipelines. For the shape optimization, we present an efficient algorithm which minimizes a non-linear non-convex functional at a fraction of the cost and with better optimality compared to standard solvers. Future research could examine the potential effects of alternating layout and shape optimization, and incorporating different materials. For example, in our stool model, the joints could use a stronger and more durable material.

Figure 5.2: When designing a foldable stool we need to (1) design the open-close mechanism, (2) make parts strong enough to hold a person, (3) and analyse the contact between the parts to ensure it locks in the desired configuration.

While Chapter 3 allows us to reinforce a single part of an object, many products are made of multiple interconnected parts or designed to interact with other objects. In many scenarios, these may experience significant forces due to contact. To understand how this interaction affect the design, in Chapter 4, we study collision and contact handling, and propose an algorithm for robust contact resolution suitable for integration into optimization pipelines. Our algorithm resolves contact using a barrier-based interior-point optimization, modified to efficiently handle a large number of contact constraints, and ensure the simulation step remain collision-free. We tested our algorithm in a set of 18 scenes with six different configurations each, showing we can robustly handle different simulations without tweaking

the optimization parameters. Our approach can be used to batch process simulations, which is an essential part of design optimization for dynamics.

One of the motivating applications for Chapter 4 is packaging. Even objects that are not designed for dynamic loading are subjected to these during distribution. During transport, it is common for packages to be dropped, or kicked; and vibration of the truck and aircraft are transmitted to the package. Additionally, the package is subject to different static loads than what the object was designed for, due to stacking or pressure. Since these loading cases happen only during distribution, it does not make sense to reinforce the object directly. Instead, we design packages that will cushion the impact and protect the product. To be able to design optimal packages efficiently, we must be able to simulate relevant scenarios (e.g., dropping a package) robustly, as a part of the optimization loop. The work described in Chapter 4 provides an important piece of a future pipeline, robust collisions resolution, but more work is required to evaluate different designs accurately and efficiently.

Finally, as we mentioned before, products are often designed to interact with other objects or incorporate existing components [37]. For example, the models in Figure 5.3.a was designed to smoothly slide 0.5in on a metallic rod and the ones in Figure 5.3.b to fit standard nuts and bolts. Designing these to "perfectly fit" usually requires multiple iterations of design and 3D printing, because of the accumulated error introduced by different tools (e.g caliper and nozzle size). To fully support customized manufacturing for these type of designs, we must understand how to best incorporate these physical constraints into the design process.

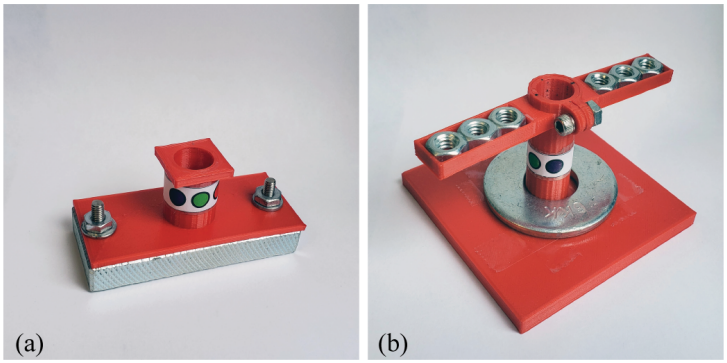


Figure 5.3: Parts in red were designed to fit with existing components: (a) the cylindric part fits a 0.5in metallic rod smoothly enough to slide but without room to tilt; (b) the hose fits a screw that is used to tighten the two parts.

Bibliography

[1] Aage, N., Andreassen, E., and Lazarov, B. S. (2015). Topology optimization using PETSc: An easy-to-use, fully parallel, open source topology optimization framework. *Structural and Multidisciplinary Optimization*, 51(3):565–572.

[2] Abdel-Malek, K., Yang, J., Blackmore, D., and Joy, K. (2006). Swept volumes: Fundation, perspectives, and applications. *International Journal of Shape Modeling*, 12(1):87–127.

[3] Aigerman, N. and Lipman, Y. (2015). Orbifold tutte embeddings. *ACM Trans. Graph.*, 34(6):190:1–190:12.

[4] Allaire, G. (2002). *Shape Optimization by the Homogenization Method*. Number v. 146 in Applied Mathematical Sciences. Springer.

[5] Allaire, G. and Jouve, F. (2008). Minimum stress optimal design with the level set method. *Engineering analysis with boundary elements*, 32(11):909–918.

[6] Allard, J., Faure, F., Courtecuisse, H., Falipou, F., Duriez, C., and Kry, P. G. (2010). Volume contact constraints at arbitrary resolution. *ACM Trans. Graph.*, 29(4):82:1–82:10.

[7] ApS, M. (2015). *The MOSEK optimization toolbox for MATLAB manual. Version 7.1 (Revision 28).*

[8] Au, O. K.-C., Tai, C.-L., and Fu, H. (2012). Multitouch gestures for constrained

transformation of 3d objects. In *Computer Graphics Forum*, volume 31, pages 651–660. Wiley Online Library.

[9] Bächer, M., Bickel, B., James, D. L., and Pfister, H. (2012). Fabricating articulated characters from skinned meshes. *ACM Transactions on Graphics (TOG)*, 31(4):47.

[10] Bächer, M., Coros, S., and Thomaszewski, B. (2015). Linkedit: interactive linkage editing using symbolic kinematics. *ACM Transactions on Graphics (TOG)*, 34(4):99.

[11] Bächer, M., Whiting, E., Bickel, B., and Sorkine-Hornung, O. (2014). Spin-it: optimizing moment of inertia for spinnable objects. *ACM Transactions on Graphics (TOG)*, 33(4):96.

[12] Bangor, A., Kortum, P., and Miller, J. (2009). Determining what individual sus scores mean: Adding an adjective rating scale. *Journal of usability studies*, 4(3):114–123.

[13] Baraff, D. (1994). Fast contact force computation for nonpenetrating rigid bodies. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, pages 23–34, New York, NY, USA. ACM.

[14] Baraff, D. (1997). An introduction to physically based modeling: rigid body simulation ii—nonpenetration constraints. *SIGGRAPH course notes*, pages D31–D68.

[15] Bendsøe, M. P. and Sigmund, O. (2004). *Topology optimization: theory, methods, and applications*. Springer Berlin Heidelberg.

[16] Berman, B. (2012). 3-d printing: The new industrial revolution. *Business horizons*, 55(2):155–162.

[17] Bommes, D., Lévy, B., Pietroni, N., Puppo, E., Silva, C., Tarini, M., and Zorin, D. (2013). Quad-mesh generation and processing: A survey. In *Computer Graphics Forum*, volume 32, pages 51–76. Wiley Online Library.

[18] Bommes, D., Zimmer, H., and Kobbelt, L. (2009). Mixed-integer quadrangulation. *ACM Transactions On Graphics (TOG)*, 28(3):77.

[19] Bowers, J., Wang, R., Wei, L.-Y., and Maletz, D. (2010). Parallel poisson disk sampling with spectrum analysis on surfaces. In *ACM Transactions on Graphics (TOG)*, volume 29, page 166. ACM.

[20] Boyd, S. and Vandenberghe, L. (2004). *Convex optimization*. Cambridge university press.

[21] Byrd, R. H., Gilbert, J. C., and Nocedal, J. (2000). A trust region method based on interior point techniques for nonlinear programming. *Mathematical Programming*, 89(1):149–185.

[22] Calì, J., Calian, D. A., Amati, C., Kleinberger, R., Steed, A., Kautz, J., and Weyrich, T. (2012). 3d-printing of non-assembly, articulated models. *ACM Transactions on Graphics (TOG)*, 31(6):130.

[23] Campen, M., Bommes, D., and Kobbelt, L. (2015). Quantized global parametrization. *j-TOG*, 34(6):192.

[24] Catto, E. (2011). Box2d: A 2d physics engine for games.

[25] Catto, E. (2018). Box2d frequently asked questions.

[26] Chen, D., Levin, D. I., Matusik, W., and Kaufman, D. M. (2017). Dynamics-aware numerical coarsening for fabrication design. *ACM Trans. Graph.*, 34(4).

[27] Coros, S., Thomaszewski, B., Noris, G., Sueda, S., Forberg, M., Sumner, R. W., Matusik, W., and Bickel, B. (2013). Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)*, 32(4):83.

[28] Da Silveira, G., Borenstein, D., and Fogliatto, F. S. (2001). Mass customization: Literature review and research directions. *International journal of production economics*, 72(1):1–13.

[29] Denavit, J. and Hartenberg, R. S. (1964). *Kinematic synthesis of linkages*. McGraw-Hill.

[30] Ebke, H.-C., Schmidt, P., Campen, M., and Kobbelt, L. (2016). Interactively controlled quad remeshing of high resolution 3d models. *ACM Trans. Graph.*, 35(6):218:1–218:13.

[31] Gal, R., Sorkine, O., Mitra, N. J., and Cohen-Or, D. (2009). iwires: an analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (TOG)*, 28(3):33.

[32] Gil-Ureta, F., Pietroni, N., and Zorin, D. (2019). Structurally optimized shells. *arXiv preprint arXiv:1904.12240*.

[33] Gil-Ureta, F., Tymms, C., and Zorin, D. (2016). Interactive modeling of mechanical objects. In *Computer Graphics Forum*, volume 35, pages 145–155. Wiley Online Library.

[34] Gilmore, J. H., Pine, B. J., et al. (1997). The four faces of mass customization. *Harvard business review*, 75(1):91–102.

[35] Grinspun, E., Gingold, Y., Reisman, J., and Zorin, D. (2006). Computing discrete shape operators on general meshes. In *Computer Graphics Forum*, volume 25, pages 547–556. Wiley Online Library.

[36] Groen, J. P. and Sigmund, O. (2017). Homogenization-based topology optimization for high-resolution manufacturable microstructures. *International Journal for Numerical Methods in Engineering*.

[37] Hague, R., Campbell, I., and Dickens, P. (2003). Implications on design of rapid manufacturing. *Proceedings of the Institution of Mechanical Engineers, Part C: Journal of Mechanical Engineering Science*, 217(1):25–30.

[38] Harmon, D., Panozzo, D., Sorkine, O., and Zorin, D. (2011). Interference-aware geometric modeling. In *ACM Transactions on Graphics (TOG)*, volume 30, page 137. ACM.

[39] Harmon, D., Vouga, E., Smith, B., Tamstorf, R., and Grinspun, E. (2009). Asynchronous contact mechanics. New York, NY, USA. ACM.

[40] Hart, C. W. (1995). Mass customization: conceptual underpinnings, opportunities and limits. *International Journal of Service Industry Management*, 6(2):36–45.

[41] Hemp, W. S. (1973). *Optimum structures*. Oxford engineering science series. Oxford, Clarendon Press.

[42] Hergel, J. and Lefebvre, S. (2015). 3d fabrication of 2d mechanisms. In *Computer Graphics Forum*, volume 34, pages 229–238. Wiley Online Library.

[43] Hessman, T. (2014). Have it your way: manufacturing in the age of mass customization. *Industry Week*, 263:14–16.

[44] Hu, Y., Zhou, Q., Gao, X., Jacobson, A., Zorin, D., and Panozzo, D. (2018). Tetrahedral meshing in the wild. *ACM Trans. Graph.*, 37(4):60:1–60:14.

[45] Jiang, C., Tang, C., Seidel, H.-P., and Wonka, P. (2017). Design and volume optimization of space structures. *ACM Transactions on Graphics*.

[46] Kälberer, F., Nieser, M., and Polthier, K. (2007). Quadcover-surface parameterization using branched coverings. In *Computer Graphics Forum*, volume 26, pages 375–384. Wiley Online Library.

[47] Kaldor, J. M., James, D. L., and Marschner, S. (2008). Simulating knitted cloth at the yarn level. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, pages 65:1–65:9, New York, NY, USA. ACM.

[48] Kaufman, D. M., Sueda, S., James, D. L., and Pai, D. K. (2008). Staggered projections for frictional contact in multibody systems. In *ACM Transactions on Graphics (TOG)*, volume 27, page 164. ACM.

[49] Kilian, M., Pellis, D., Wallner, J., and Pottmann, H. (2017). Material-minimizing forms and structures. *ACM Transactions on Graphics (TOG)*, 36(6):173.

[50] Knöppel, F., Crane, K., Pinkall, U., and Schröder, P. (2015). Stripe patterns on surfaces. *j-TOG*, 34(4):39:1–39:11.

[51] Koo, B., Li, W., Yao, J., Agrawala, M., and Mitra, N. (2014). Creating works-like prototypes of mechanical objects. *ACM Transactions on Graphics*, 33(6).

[52] Koyama, Y., Sueda, S., Steinhardt, E., Igarashi, T., Shamir, A., and Matusik, W. (2015). Autoconnect: computational design of 3d-printable connectors. *ACM Transactions on Graphics (TOG)*, 34(6):231.

[53] Kraft, D. (1988). A software package for sequential quadratic programming. *Forschungsbericht- Deutsche Forschungs- und Versuchsanstalt fur Luft- und Raumfahrt.*

[54] Lampel, J. and Mintzberg, H. (1996). Customizing customization. *Sloan management review*, 38(1):21–30.

[55] Lau, M., Ohgawara, A., Mitani, J., and Igarashi, T. (2011). Converting 3d furniture models to fabricatable parts and connectors. In *ACM Transactions on Graphics (TOG)*, volume 30, page 85. ACM.

[56] Li, H., Hu, R., Alhashim, I., and Zhang, H. (2015). Foldabilizing furniture. *ACM Transactions on Graphics (TOG)*, 34(4):90.

[57] Li, W., Zheng, A., You, L., Yang, X., Zhang, J., and Liu, L. (2017). Rib-reinforced shell structure. In *Computer Graphics Forum*, volume 36, pages 15–27. Wiley Online Library.

[58] Li, Y. and Chen, Y. (2010). Beam structure optimization for additive manufacturing based on principal stress lines. In *Solid Freeform Fabrication Proceedings*, pages 666–678.

[59] Lu, L., Rahimian, A., and Zorin, D. (2017). Contact-aware simulations of particulate stokesian suspensions. *Journal of Computational Physics*, 347:160–182.

[60] Lu, L., Sharf, A., Zhao, H., Wei, Y., Fan, Q., Chen, X., Savoye, Y., Tu, C., Cohen-Or, D., and Chen, B. (2014). Build-to-last: Strength to weight 3d printed objects. *ACM Transactions on Graphics (TOG)*, 33(4):97.

[61] Mallik, A. K., Ghosh, A., and Dittrich, G. (1994). *Kinematic analysis and synthesis of mechanisms*. CRC Press.

[62] Megaro, V., Thomaszewski, B., Nitti, M., Hilliges, O., Gross, M., and Coros, S. (2015). Interactive design of 3d-printable robotic creatures. *ACM Transactions on Graphics (TOG)*, 34(6):216.

[63] Michell, A. (1904). The limits of economy of material in frame-structures. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 8(47):589–597.

[64] Mitra, N. J., Yang, Y.-L., Yan, D.-M., Li, W., and Agrawala, M. (2010). Illustrating how mechanical assemblies work. *ACM Transactions on Graphics (TOG)*, 29(4):58.

[65] Müller, M., Chentanez, N., Kim, T.-Y., and Macklin, M. (2015). Air meshes for robust collision handling. *ACM Transactions on Graphics (TOG)*, 34(4):133.

[66] Munk, D. J., Vio, G. A., and Steven, G. P. (2015). Topology and shape optimization methods using evolutionary algorithms: a review. *Structural and Multidisciplinary Optimization*, 52(3):613–631.

[67] Myles, A., Pietroni, N., and Zorin, D. (2014a). Robust field-aligned global parametrization. *ACM Trans. Graph.*, 33(4).

[68] Myles, A., Pietroni, N., and Zorin, D. (2014b). Robust field-aligned global parametrization. *ACM Transactions on Graphics (TOG)*, 33(4):135.

[69] Oñate, E., Zárate, F., and Flores, F. (1994). A simple triangular element for thick and thin plate and shell analysis. *International Journal for Numerical Methods in Engineering*, 37:2569.

[70] Otaduy, M. A., Tamstorf, R., Steinemann, D., and Gross, M. (2009). Implicit contact handling for deformable objects. In *Computer Graphics Forum*, volume 28, pages 559–568. Wiley Online Library.

[71] Panetta, J., Zhou, Q., Malomo, L., Pietroni, N., Cignoni, P., and Zorin, D. (2015). Elastic textures for additive fabrication. *ACM Transactions on Graphics (TOG)*, 34(4):135.

[72] Peternell, M., Pottmann, H., Steiner, T., and Zhao, H. (2005). Swept volumes. *Computer-Aided Design and Applications*, 2(5):599–608.

[73] Pietroni, N., Tonelli, D., Puppo, E., Froli, M., Scopigno, R., and Cignoni, P. (2015). Statics aware grid shells. In *Computer Graphics Forum*, volume 34, pages 627–641. Wiley Online Library.

[74] Pine, B. J. (1993). Mass customizing products and services. *Planning review*, 21(4):6–55.

[75] Prévost, R., Whiting, E., Lefebvre, S., and Sorkine-Hornung, O. (2013). Make it stand: balancing shapes for 3d fabrication. *ACM Transactions on Graphics (TOG)*, 32(4):81.

[76] Ray, N. and Sokolov, D. (2014). Robust polylines tracing for n-symmetry direction field on triangulated surfaces. *ACM Transactions on Graphics (TOG)*, 33(3):30.

[77] Redon, S., Kheddar, A., and Coquillart, S. (2002). Fast continuous collision detection between rigid bodies. In *Computer graphics forum*, volume 21, pages 279–287. Wiley Online Library.

[78] Rozvany, G. I. (1976). *Optimal design of flexural systems: beams, grillages, slabs, plates and shells*. Elsevier.

[79] Rozvany, G. I. (2012). *Structural design via optimality criteria: the Prager approach to structural optimization*, volume 8. Springer Science & Business Media.

[80] Schneider, T., Dumas, J., Gao, X., Zorin, D., and Panozzo, D. (2019). Polyfem. https://polyfem.github.io/.

[81] Schulz, A., Shamir, A., Levin, D. I., Sitthi-Amorn, P., and Matusik, W. (2014). Design and fabrication by example. *ACM Transactions on Graphics (TOG)*, 33(4):62.

[82] Sigmund, O. (2001). A 99 line topology optimization code written in matlab. *Structural and multidisciplinary optimization*, 21(2):120–127.

[83] Sigmund, O., Aage, N., and Andreassen, E. (2016). On the (non-) optimality of michell structures. *Structural and Multidisciplinary Optimization*, 54(2):361–373.

[84] Sigmund, O. and Maute, K. (2013). Topology optimization approaches. *Structural and Multidisciplinary Optimization*, 48(6):1031–1055.

[85] Smith, J., Hodgins, J., Oppenheim, I., and Witkin, A. (2002). Creating models of truss structures with optimization. *ACM Trans. Graph.*, 21(3):295–301.

[86] Sokół, T. (2011). A 99 line code for discretized michell truss optimization written in mathematica. *Structural and Multidisciplinary Optimization*, 43(2):181–190.

[87] Sokół, T. and Rozvany, G. I. (2016). A new adaptive ground structure method for multi-load spatial michell structures. *Advances in mechanics: Theoretical, computational and interdisciplinary issues*, pages 525–528.

[88] Spira, J. S. (1993). Mass customization through training at lutron electronics. *Planning Review*, 21(4):23–24.

[89] Stewart, D. E. and Trinkle, J. C. (1996). An implicit time-stepping scheme for rigid body dynamics with inelastic collisions and coulomb friction. *International Journal for Numerical Methods in Engineering*, 39(15):2673–2691.

[90] Strang, G. and Kohn, R. V. (1983). Hencky-prandtl nets and constrained michell trusses. *Computer Methods in Applied Mechanics and Engineering*, 36(2):207–222.

[91] Tam, K.-M. M. (2015). *Principal stress line computation for discrete topology design*. PhD thesis, Massachusetts Institute of Technology.

[92] Tam, K.-M. M., Coleman, J. R., Fine, N. W., and Mueller, C. T. (2015). Stress line additive manufacturing (slam) for 2.5-d shells.

[93] Thomaszewski, B., Coros, S., Gauge, D., Megaro, V., Grinspun, E., and Gross, M. (2014). Computational design of linkage-based characters. *ACM Transactions on Graphics (TOG)*, 33(4):64.

[94] Thompson, M. K., Moroni, G., Vaneker, T., Fadel, G., Campbell, R. I., Gibson, I., Bernard, A., Schulz, J., Graf, P., Ahuja, B., et al. (2016). Design for additive manufacturing: Trends, opportunities, considerations, and constraints. *CIRP annals*, 65(2):737–760.

[95] Umetani, N., Igarashi, T., and Mitra, N. J. (2012). Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*, 31(4):86.

[96] Vaxman, A., Campen, M., Diamanti, O., Panozzo, D., Bommes, D., Hildebrandt, K., and Ben-Chen, M. (2016). Directional field synthesis, design, and processing. In *Computer Graphics Forum*, volume 35, pages 545–572. Wiley Online Library.

[97] von Dziegielewski, A. and Hemmer, M. (2011). High quality surface mesh generation for swept volumes. In *EuroCG Workshop on Computational Geometry*.

[98] Wu, J., Dick, C., and Westermann, R. (2016). A system for high-resolution topology optimization. *IEEE Transactions on Visualization and Computer Graphics*, 22(3):1195–1208.

[99] Xu, W., Wang, J., Yin, K., Zhou, K., Van De Panne, M., Chen, F., and Guo, B. (2009). Joint-aware manipulation of deformable models. In *ACM Transactions on Graphics (TOG)*, volume 28, page 35. ACM.

[100] Zegard, T. and Paulino, G. H. (2014). Grand — ground structure based topology optimization for arbitrary 2d domains using matlab. *Structural and Multidisciplinary Optimization*, 50(5):861–882.

[101] Zegard, T. and Paulino, G. H. (2015). Grand3 — ground structure based topology optimization for arbitrary 3d domains using matlab. *Structural and Multidisciplinary Optimization*, 52(6):1161–1184.

[102] Zegard, T. and Paulino, G. H. (2016). Bridging topology optimization and additive manufacturing. *Structural and Multidisciplinary Optimization*, 53(1):175–192.

[103] Zhao, H., Xu, W., Zhou, K., Yang, Y., Jin, X., and Wu, H. (2017). Stress-constrained thickness optimization for shell object fabrication. In *Computer Graphics Forum*, volume 36, pages 368–380. Wiley Online Library.

[104] Zhou, Q., Grinspun, E., Zorin, D., and Jacobson, A. (2016a). Mesh arrangements for solid geometry. *ACM Transactions on Graphics (TOG)*, 35(4).

[105] Zhou, Q., Grinspun, E., Zorin, D., and Jacobson, A. (2016b). Mesh arrangements for solid geometry. *ACM Trans. Graph.*, 35(4):39:1–39:15.

[106] Zhou, Q., Panetta, J., and Zorin, D. (2013). Worst-case structural analysis. *ACM Transactions on Graphics (TOG)*, 32(4):137.

[107] Zhu, L., Xu, W., Snyder, J., Liu, Y., Wang, G., and Guo, B. (2012). Motion-guided mechanical toy modeling. *ACM Trans. Graph.*, 31(6):127.