The constrained geometry of structures: Optimization methods for inverse form-finding design

by

Pierre Cuvilliers

Diplôme de l'École polytechnique, 2014
Ingénieur de l'École nationale des ponts et chaussées, 2016

Submitted to the Department of Architecture
in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy in Architecture: Building Technology

at the

Massachusetts Institute of Technology

May 2020

Signature of Author: ____

Department of Architecture
May 1, 2020

Certified by: ____

Caitlin T. Mueller
Associate Professor of Architecture and Civil and Environmental Engineering
Thesis Supervisor

Accepted by: ____

Leslie K. Norford
Professor of Building Technology
Chair, Department Committee on Graduate Students

Dissertation committee

Caitlin T. Mueller
Associate Professor of Architecture and Civil and Environmental Engineering
Massachusetts Institute of Technology
Thesis Supervisor

Sigrid Adriaenssens
Associate Professor of Civil and Environmental Engineering
Princeton University
Thesis Reader

John A. Ochsendorf
Professor of Architecture and Civil and Environmental Engineering
Massachusetts Institute of Technology
Thesis Reader

The constrained geometry of structures: Optimization methods for inverse form-finding design

by

Pierre Cuvilliers

Submitted to the Department of Architecture on May 1, 2020 in Partial Fulfillment of the Requirements for the Degree of Doctor of Philosophy in Architecture: Building Technology

Abstract

This dissertation aims to improve form-finding workflows by giving more control on the obtained shapes to the designer. Traditional direct form-finding allows the designer to generate shapes for structures that need to verify a mechanical equilibrium when built; however, it produces shapes that are difficult to control. This dissertation shows how the design of constrained structural systems is better solved by an inverse form-finding process, where the parameters and initial conditions of the direct form-finding process are automatically adjusted to match the design intent. By defining a general framework for the implementation of such workflows in a nested optimizer loop, the requirements on each component are articulated. The inner optimizer is a specially selected direct form-finding solver, the outer optimizer is a general-purpose optimization routine. This is demonstrated with case studies of two structural systems: bending-active structures and funicular structures. These two systems that can lead to efficient covering structures of long spans.

For bending-active structures, the performance (speed, accuracy, reliability) of direct form-finding solvers is measured. Because the outer optimization loop in an inverse form-finding setup needs to rely on a robust forward simulation with minimal configuration, we find that general-purpose optimizers like SLSQP and L-BFGS perform better than domain-specific algorithms like dynamic relaxation. Using this insight, an inverse form-finding workflow is built and applied with a closest-fit optimization objective.

In funicular structures, this dissertation first focuses on a closest-fit to target surface optimization, giving closed-form formulations of gradients and hessian of the problem. Finding closed-form expressions of these derivatives is a major blocking point in creating more versatile inverse form-finding workflows. This process optimizer is then reimplemented in an Automatic Differentiation framework, to produce an inverse form-finding tool for funicular surfaces with modular design objectives. This is a novel way of implementing such tools, exposing how the design intent can be represented by more complex objects than a target surface. Reproducing existing structures, and generating more efficient funicular shapes for them, the possibilities of the tool are demonstrated in exploring the design space and fine-tuned modifications, thanks to the fine control over the objectives representing the design intent.

Keywords: form-finding, inverse problems, non-linear optimization, computational design

Thesis supervisor: Caitlin T. Mueller

Title: Associate Professor of Architecture and Civil and Environmental Engineering

Acknowledgments

I am very grateful to everyone that helped me on the way to complete this dissertation: advisors, colleagues, friends, and family who all contributed to make this journey an amazing experience.

First, I would like to thank my advisor, Professor Caitlin Mueller, for her always insightful and patient guidance and support throughout these five years. Her deep knowledge of the computational design of structures was always shared with passion, and pushed me to take a new look on my work as an engineer. She has been an amazing guide in becoming a better teacher and researcher in our field, helping me find balance between precision, creativity, and impact.

I am also very thankful to Sigrid Adriaenssens and John Ochsendorf, the two other members of my dissertation committee. Their comments and guidance have had an enormous impact on this dissertation and defense. They both shared their long built knowledge of bending-active structures and funicular shells without restrictions and helped me build a strong argument to convince experts such as them. The other faculty of the Building Technology lab – John Fernández, Leon Glicksman, Les Norford, and Christoph Reinhart, also deserve many thanks for creating a true multi-disciplinary research environment, where I could learn so much. I am also grateful for the MIT presidential fellowship for providing me with continued financial support throughout my dissertation.

I was helped in my work for this dissertation by many colleagues, to whom I owe a great deal. Toby Mitchell shared his very precise knowledge of the force density method and related tools, and reminded me of how inverse form-finding design, and all the gradients needed for it, were mentioned in Schek's work as early as 1974. Justina Yang worked with me as an undergraduate research assistant during my first year at MIT, and built very useful tools to run many of the simulations of Chapter 3. Renaud Danhaive was my classmate in our early years at MIT and helped me define many of the early ideas for this thesis. I thoroughly enjoyed working with him on several research projects, such as the initial version of Chapter 5. Paul Mayencourt convinced me to get my tools used by many more than I was initially comfortable with, and was a great teaching partner for the AAG workshop of Chapter 4. I am also very grateful to all the students of the Digital Structures research group, and the Building Technology lab, for their friendship, support, and collaborative spirit.

A special thank you goes to the makers of Rhino3D at McNeel and associates, who were my colleagues for two summer internships and helped me tremendously in becoming a better programmer. In particular, thank you to Bob McNeel, who organized excellent conditions each time and also agreed to fund the final year of my Ph.D., and to Greg Arden, Dale Lear and Chuck Welsh who taught me most of what I know about NURBS.

Finally, a big thank you to all my family and friends, here in the US and back home in France, who gave their love, support and constant encouragement throughout my Ph.D., despite the distance. Without you, I could not have done it.

À Alizée, merci.

Table of Contents

List of Figures

List of Tables

Publications related to this dissertation

Chapter 3 was adapted from:

Cuvilliers, P., & Mueller, C. (2018, July). A comparison of dynamic relaxation and generalist optimization methods for the simulation of bending-active structures. *Proceedings of the IASS Symposium 2018*. IASS Symposium 2018, Boston, MA, USA,

and:

Cuvilliers, P., Yang, J. R., Coar, L., & Mueller, C. (2018). A comparison of two algorithms for the simulation of bending-active structures. *International Journal of Space Structures*, *33*(2), 73–85. https://doi.org/10.1177/0266351118779979.

The methods presented in Chapter 4 were also taught as:

P. Cuvilliers, P. Mayencourt, and C. Mueller, "Design and fabrication of bending-active structures with controlled shapes: the arc lamp," in *Workshops of the Advances in Architectural Geometry Conference*, Gothenburg, Sweden.

Chapter 5 was adapted from:

Cuvilliers, P., Danhaive, R., & Mueller, C. T. (2016). Gradient-based optimization of closest-fit funicular structures. In K. Kawaguchi, M. Ohsaki, & T. Takeuchi (Eds.), *Spatial Structures: Proceedings of the IASS Symposium 2016.*

The contents of Chapter 6 were presented as:

Cuvilliers, P., & Mueller, C. (2019, October). *Differentiable force-density method: An easily extensible framework for the constrained design of funicular shape*s. IASS International Symposium 2019, Barcelona, Spain.

Most of the design tools presented in this dissertation use a specially developed connection between Rhino3d and Python, released as open-source software:

Cuvilliers, P., & Mueller, C. (2017). *GH Python Remote*. github.com/Digital-Structures/ghpythonremote.

1 Introduction

1.1 Problem statement

Building for freeform architecture, to create structures that physically realize the shape of a complex surface, introduces challenges never encountered when building orthogonal structures. The choices made on the construction process and structural quality of the building will dictate constraints for the surface. Additionally, while matching these constraints, the designer usually follows an architectural intent: for example a target shape to fit as well as possible. Small examples of this process are represented in Figure 1.1, where different construction systems are matched with three target surfaces. Each construction system leads to different constraints on the constructible surface, generating different final shapes, and it is impossible initially to predict how these shapes will differ nor how to layout a specific construction system to get as close as possible to the target.

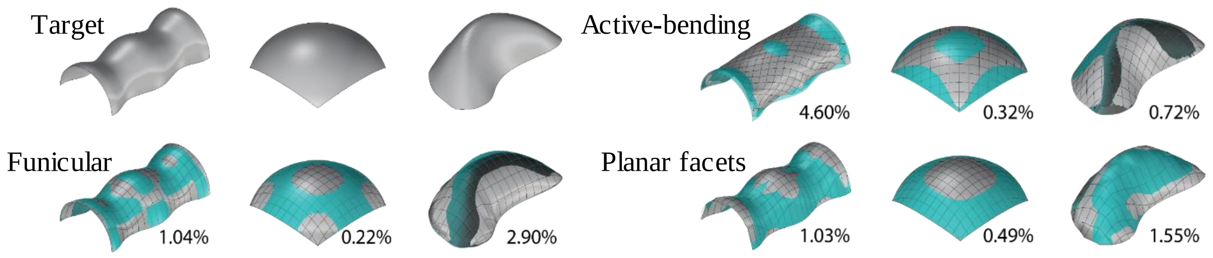


Figure 1.1: Three architectural shell shapes (grey, top left) are approximated using different construction systems (green). Each construction system leads to different constraints on the constructible surface, generating different final shapes. The score values given are calculated as the RMS of the distances d_t from each vertex t to the closest point on the target surface, normalized by the nominal length of the mesh edges: $\sqrt{\sum_t d_t^2}/n \times size_{cell}$

1.1.1 Structural systems

This dissertation uses the term *construction system* as a way to distinguish between structural systems that can be used to build free-form surfaces. It focuses specifically on two structural systems, for which it will present methods to generate viable shapes. First, bending-active structures (see Chapter 3 for images), are made of long rods of timber or fiberglass, that are bent into shape and undergo large deformations in this process. This dissertation focuses in particular on elastic gridshell, bending-active structures that start from a flat regular grid of such rods, pinned at all intersecting nodes, that is then bent and to the ground attached on its edge. They make efficient, lightweight covers that are quick to build. Simulating the deformation process of the grid is the only way to know what the final shape will be. This means that to find a viable shape for elastic gridshells, that can be built with such a structural system, simulating this deformation process on one specific instance of elastic gridshell is necessary.

The second system is funicular structures, such as thin concrete shells that behave in compression-only (see Chapter 5 for images). They are another efficient way of covering large spaces with minimal material and many were built in the mid-twentieth century. To remain thin as they are built, these structures need to have minimal bending stresses in them. This means viable shapes for thin shell will be funicular against their main load (most often their self-weight), which means that this loading condition will only generate membrane compression forces in the shape.

1.1.2 Designer control

This dissertation then attempts to give designers as much control as possible when they are generating shapes for these structural systems. This control can often be expressed through an objective to achieve, such as a target shape that the generated viable shape should approximate. The target can also be the result of another process that generates structurally efficient shapes. For example, in Figure 1.2, a concrete shell is cast on top of an elastic gridshell. The concrete shell shape is the result of a form-finding process that guarantees compression-only forces, and the elastic gridshell has to be designed to get as close as possible to this target shape while matching its own set of constraints.

In both cases, this intent needs to be realized with a structural system, that introduces geometrical constraints. Designers need the tools that will find one such realization and let them explore new shapes around this solution. Two paths have traditionally been taken to design freeform surfaces with a physical reality. The first one is to only draw surfaces that match some global constraints, like drawing developable surfaces to design a roof built with continuous metal sheets. This is possible with good accuracy and speed in CAD software, but this process only works up to the extent of one continuous developable patch. The second path taken is the post-rationalization of the freeform design. There, the designer draws with the software's capabilities as the only constraint, and then uses a black-box to find a similar surface that can be built using the selected technology. This process is often frustrating for the designer, who does not have control over what the result of their drawing will be and gets limited feedback on which inputs should be modified to change this result.

More generally, none of these paths let the designer explore and optimize in the design space that the construction system defines, because they do not have a concept to represent this design space. Yet this *inverse form-finding* problem, where the goal is to get an instance of a construction system as close as possible to a target shape, is ubiquitous in architecture. The designer has a shape in mind, knows what system to use to build it, but cannot predict how and what to lay out to achieve it. This proposal defines a category of computational tools needed to solve these problems, and offers to create them.

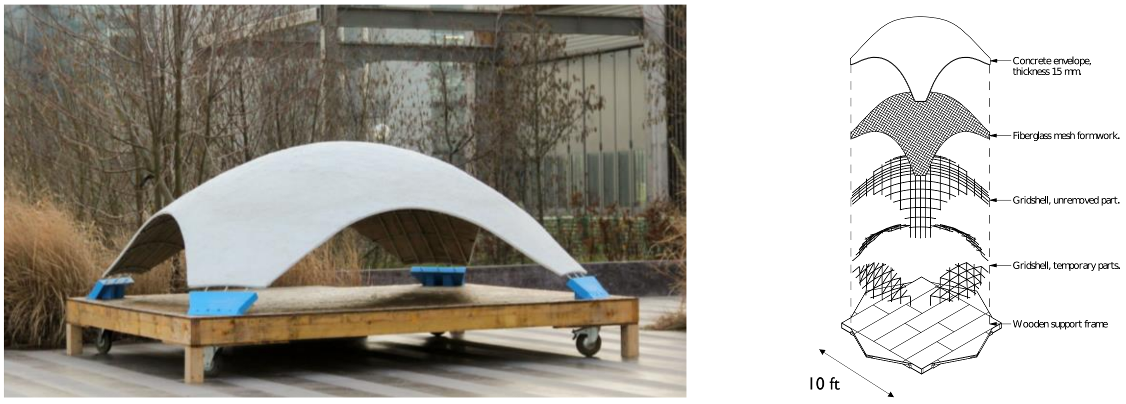

Concrete envelope, thickness 15 mm.
Fiberglass mesh formwork.
Gridshell, unremoved part.
Gridshell, temporary parts.
Wooden support frame
10 ft

Figure 1.2: Concrete shell on elastic gridshell. In this system, the shape of the elastic gridshell must match the geometric constraints of elastic gridshells and be as close as possible to a funicular shape for the concrete shell. (Cuvilliers et al., 2017)

1.2 Research motivation and purpose

In building inverse form-finding tools, we hope to give the designer more control on the shapes of complex structural systems, that are governed by specific rules limiting their design spaces. Because these shapes adapt to bespoke construction systems, they can be chosen to use material very efficiently. Funicular shapes for example, will very efficiently span large spaces. Bending-active structures are very adaptable and have a large potential for reuse. Additionally, thanks to this added control, designers can explore larger regions of the design space of their chosen construction system, leading to more variety in structures without any penalty on efficiency. While the shapes of many shells were initially chosen as pre-defined mathematical functions like revolution surfaces of lines or simple curves, so that their analysis would be simpler, we now have tools to create and analyze free-form funicular shells.

This new control on the form-finding process will also give designers more freedom in how they use these tools. While existing direct form-finding tools tend to give results of a similar architectural style, designers can now guide the exploration towards target shapes and author their own style on them.

A motivating example can be made for this research based on the design process of the Viadotto dell'Industria, or bridge on the Basento river, designed by architect and engineer Sergio Musmeci and built in 1976 in Potenza, Italy. The bridge is a road bridge where the deck is supported at regular intervals by a concrete shell spanning over the river and other pathways in the valley below it, see Figure 1.3. In the design of the bridge, Musmeci used physical and analytical models to form-find the shape of a membrane that would be tied to the ground anchors, and the deck support positions (Marmo et al., 2019). This generates a shape that spans from the ground anchors to the deck supports, but is not funicular under its own weight and the dead loads of the bridge. That leads to an inefficient structure as the shell will have to support bending forces in this principal loading state. Multiple optimizations were made on the position of the supports, and the shape of the form-found membrane-like shell, but still, in the final result the shell is much thicker than can be achieved with a true funicular shape.

A better way to find a shape for the shell is to use a form-finding process that generates only funicular shapes, like the force density method (see Chapter 5). However, this process does not guarantee the position of the deck supports in the form-found shape (Figure 1.4, top left): these cannot be described as fixed anchor points otherwise additional forces would be introduced in the shell, moving it away from funicularity. Instead, parameters of the form-finding process can be adjusted so that the support points get closer to the deck. Doing this manually is very tedious, and almost impossible (Figure 1.4, top right and bottom left): in the force density method, there is one independent parameter that can be chosen for each edge of the simulation mesh, and changing each one of them can affect the whole form-found shape. Instead, an automatic way of changing these parameters is needed, so that a funicular shape with the support points touching the deck can be generated. Implementing such a method will be the object of Chapter 6; Figure 1.4, bottom right shows the results that can be obtained with it.

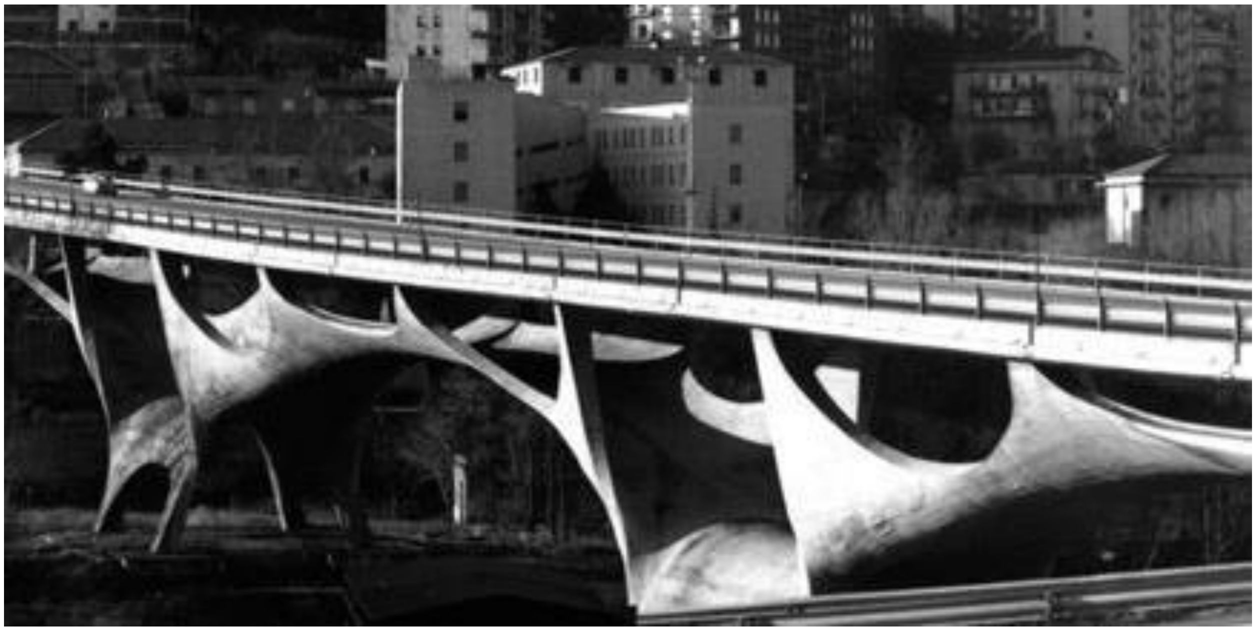


Figure 1.3: Viadotto dell'Industria, 1976, Potenza, Italy. Arch./Eng. Sergio Musmeci. © Alba Fermoso Tango.

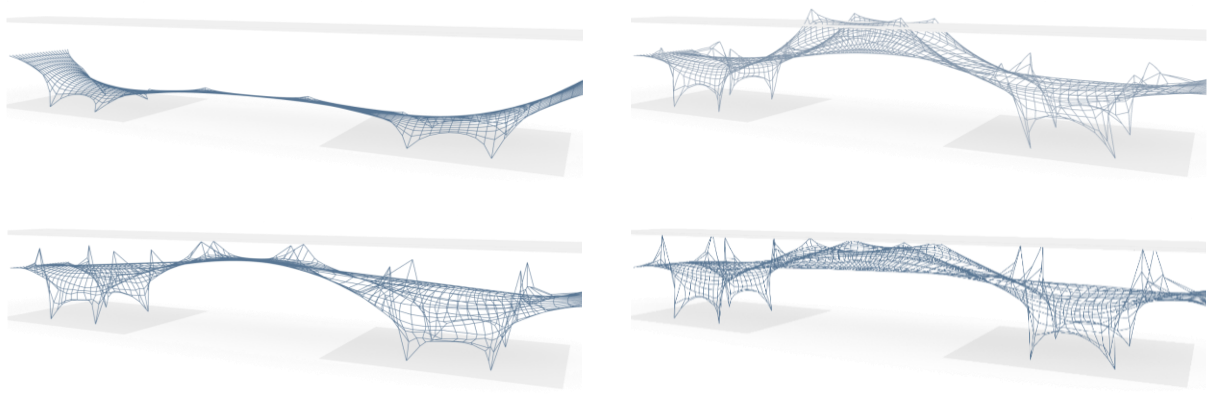


Figure 1.4: Finding a funicular shape for the Musmeci bridge, where the support points for the deck reach their desired positions. All shapes are generated using the force density method, which guarantees funicularity against the self-weight of the shell and the dead loads of the bridge at the support points. The first three images are generated using manual modifications of the force density method parameters, one for each edge in the simulation mesh; it is almost impossible to reach the deck this way. The last image (bottom right) is generated using the methods of Chapter 6, and finds a funicular shape with the prescribed positions for the deck supports.

Note that solving this exact problem, of moving points in a force density method form-finding process towards target locations, is solved by the existing extended force density method (Miki & Kawaguchi, 2010). However, there are other design constraints that the designers will want to accommodate with the results of form-finding processes, and these processes will not always be the force density method. This dissertation will attempt to solve such broad problems.

1.3 Research objective

Many of the terms used in computational design for architecture are not consistently defined, especially when looking at systems that have a structural objective. This section first lays out the definitions used in this proposal, then defines the research objective.

Form-finding is the name given to processes to generate shapes that realize some structural objective or equilibrium constraint– examples include networks of hanging chains, the fins and tensioned fabric of an umbrella, or sheet metal elastically deformed. This is a non-trivial process when the structural system considered undergoes large displacements, leading to a (geometrically) non-linear problem, usually requiring an iterative procedure to solve computationally.

It is useful to distinguish three classes of form-finding problems. First, *direct problems*: finding one physical shape under a given set of mechanical properties and boundary conditions. Often, this goes against the intuitive design process where the designer has a shape in mind and wants to minimize the difference to that target. That describes an *inverse problem* where the closest possible physical system to a target shape is found. A third, intermediate, category can be defined: *pseudo-inverse form-finding*. In this case like in inverse form-finding, the objective is to match a construction system to a target shape, but in this case the constraints are not matched exactly. These definitions are summed up in Figure 2.3.

There exist tools for solving direct form-finding problems, generating admissible instances of the construction system, but the inverse form-finding tools needed to get close to the design intent are less common. Additionally, creating tools to explore the design space around an optimal solution would offer a way to guide the designer's hand to new shapes, yet without limiting it more than the construction system would.

This research aims at creating better tools for the design of freeform surfaces with buildability constraints, effectively letting the designer formulate an inverse form-finding problem and interact with it. Two concurrent approaches are taken. One iteratively simulates physically-based direct form-finding problems and optimizes the initial conditions of this problem until the form-found shape is close to the target shape from the designer. As no closed-form solution usually exists even for the direct problem, this results in nested optimization problems. The other approach rewrites the inverse problem using special properties of a given construction system to solve it using only one level of optimization.

Generally, a mathematical optimization approach is taken. Equation (1.1) gives the general expression of an inverse form-finding problem: minimize an objective function f of the equilibrium position $\boldsymbol{x}$, where the equilibrium position is defined as the mechanical equilibrium minimizing an energy $E_{\boldsymbol{p}}$, over the variables $\boldsymbol{p}$ that are the variable parameters to the direct form-finding problem:

$$(P_{1.1})\colon \min_{\boldsymbol{p}} f(\boldsymbol{x}), \text{ subject to } \boldsymbol{x} = \operatorname{argmin}_{\mathbf{y}} E_{\boldsymbol{p}}(\boldsymbol{y}). \tag{1.1}$$

The constraints part of this optimization problem is what represents the construction system; the objective represents how well the intent is matched. Often, the objective will be to find the closest fit to a continuous target surface. Equation (1.1), mostly through the expression of the constraints, assumes that the construction system and its instances can be represented as a discrete mesh: a collection of vertices linked by edges, forming faces. For more complex construction systems that do not have well-defined nodes, like membranes, a finite element representation allows one to fall back to the mesh-like formulation, and at the same time is often necessary to simulate the behavior of the construction system.

To solve this optimization problem, this dissertation defines a nested optimization framework represented in Figure 1.5 and Figure 1.6. On the outer level, a general-purpose optimizer will explore the design space of solutions to a direct form-finding problem, until a minimum of the objective function is found. On the inner level, a specialized solver for direct form-finding problems is used to generate equilibrium positions for the given structural system. The two levels are linked through the parameters $\boldsymbol{p}$ that define a specific instance of the direct form-finding problem.

The objective of this dissertation is then to define a general framework for inverse form-finding, considering the problem set forward by Equation (1.1), and create computational tools to solve it in specific instances. This means realizing a good modularity of the building constraints and the designer's intent expressions while being fast enough to allow iterative design exploration. Good reliability also needs to be achieved in simulating construction systems. This will be done by implementing a design tool around carefully selected solvers for the optimization problem of Equation (1.1), iteratively calling on efficient direct form-finding solvers.

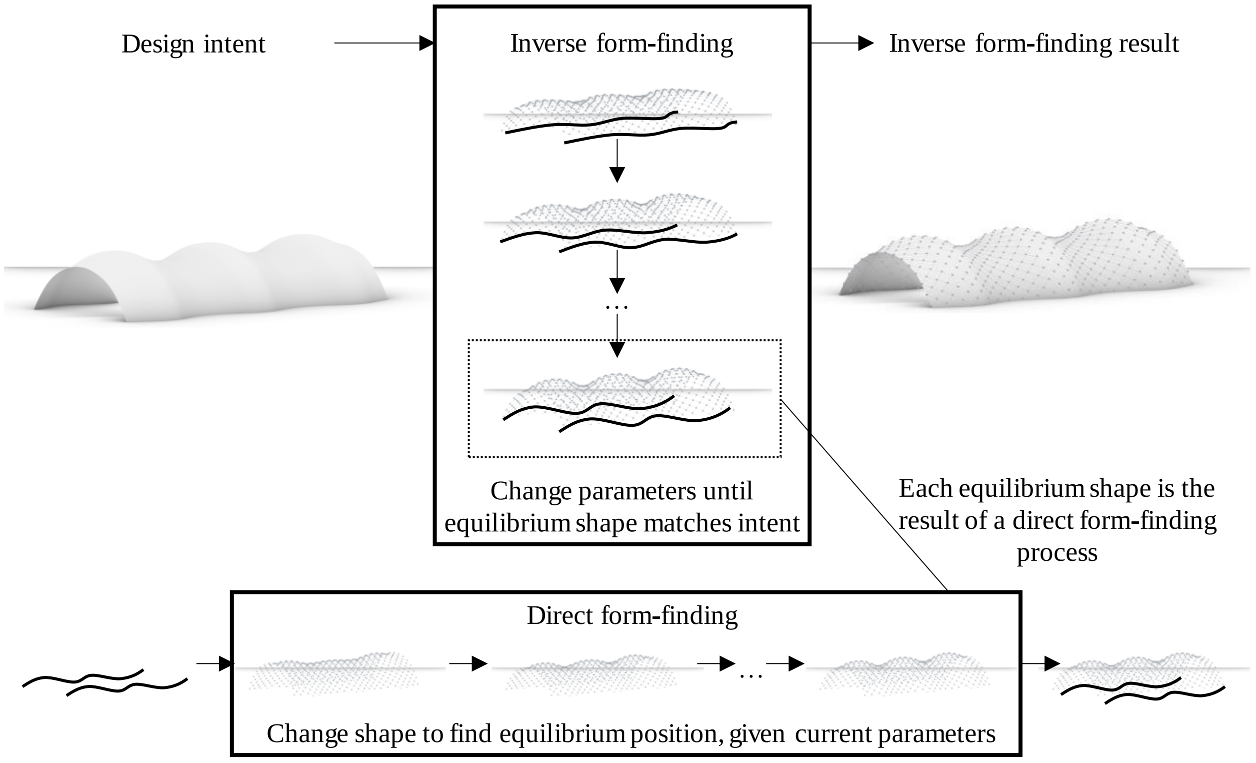

Design intent
Inverse form-finding
Inverse form-finding result
Change parameters until equilibrium shape matches intent
Each equilibrium shape is the result of a direct form-finding process
Direct form-finding
Change shape to find equilibrium position, given current parameters

Figure 1.5: Nested optimization loops for solving inverse form-finding problems, graphical view.

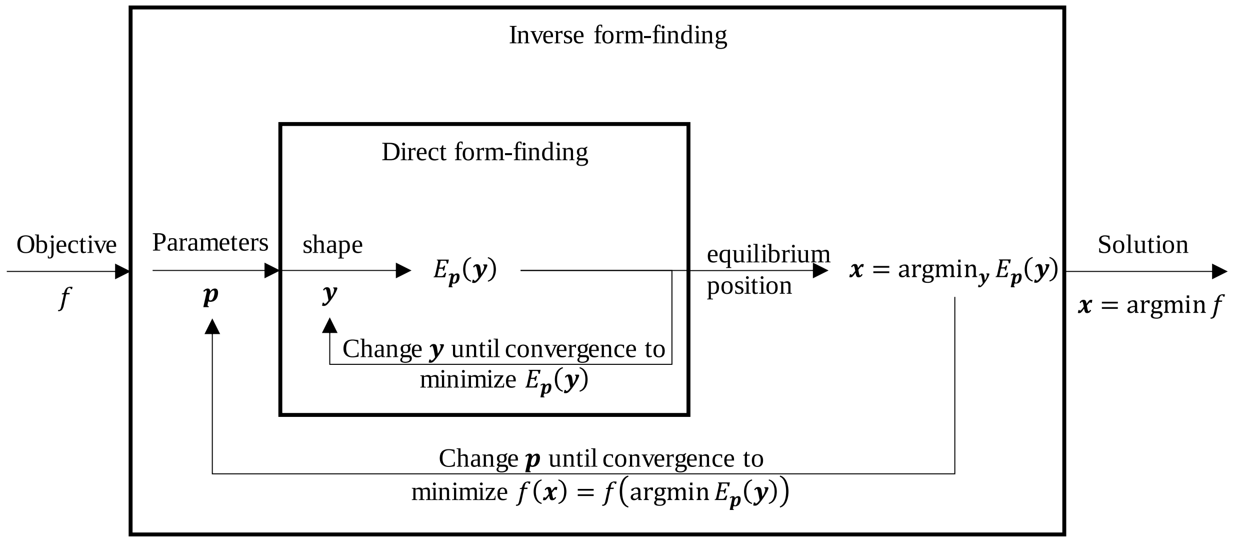

Inverse form-finding
Direct form-finding
Objective
f
Parameters
$\boldsymbol{p}$
shape
$\boldsymbol{y}$
$E_{\boldsymbol{p}}(\boldsymbol{y})$
Change $\boldsymbol{y}$ until convergence to minimize $E_{\boldsymbol{p}}(\boldsymbol{y})$
equilibrium position
$\boldsymbol{x} = \mathrm{argmin}_{\boldsymbol{y}} E_{\boldsymbol{p}}(\boldsymbol{y})$
Solution
$\boldsymbol{x} = \mathrm{argmin}\, f$
Change $\boldsymbol{p}$ until convergence to minimize $f(\boldsymbol{x}) = f\big(\mathrm{argmin}\, E_{\boldsymbol{p}}(\boldsymbol{y})\big)$

Figure 1.6: Nested optimization loops for solving inverse form-finding problems, mathematical view.

1.4 Organization of the dissertation

This dissertation aims at implementing inverse form-finding workflows for two structural systems, bending-active structures and funicular structures. Additionally, we want to gain a better understanding of how the systems can be made fast and robust. Finally, in implementing our solutions, we are looking for insights on how general frameworks for inverse form-finding could be realized. There are two main parts to this dissertation, one for each mechanics-based constrained system of bending-active structures in Chapters 3 and 4, and funicular structures in Chapters 5 and 6.

Generally, we are not concerned with the implementation of direct form-finding solvers as they are now readily available for many systems; however, for bending-active structures we will see that the very specific requirements of our system lead to re-evaluating existing direct form-finding methods. This will be done early in the dissertation (Chapter 3) and will build on the challenges of speed, reliability and accuracy for the direct form-finding solvers in the later chapters.

Chapter 2 looks at existing literature on computational design tools for construction systems with complex geometrical constraints, that let the designer express an intent to match.

Chapter 3 looks at the selection and possible improvements of direct form-finding tools for bending-active structures. Fast and reliable form-finding tools will be of great importance for Chapter 4, where they will be used repeatedly in an optimization loop – without any real-time control possible by the designer on the form-finding process – to find bending-active structures that are closest to an objective shape. We will be able to improve their implementation and gain a better understanding of how they work and perform.

In Chapter 4, we look at the inverse form-finding problem of finding a bending-active structure that is as close as possible to a target shape, for simple elastica-like shapes and elastic gridshells. This is done using the nested optimization process described earlier. In the outer optimizer, we iterate on some initial conditions of the bending-active structure, until the distance from the equilibrium shape given the current initial conditions is as small as possible. Finding the equilibrium position is the result of the inner optimizer, we build on the results of Chapter 3 for this. This inner optimizer needs to be stable, reliable and fast, given the wide variety of input conditions that the outer optimizer will attempt. We also look at finding good representations of the variable initial conditions and parameters sent by the outer optimizer to the inner one on different case studies.

Chapter 5 changes construction systems to look at funicular structures. Although they have seen more interest in terms of the inverse form-finding tools developed for them, they will prove to be a good model system for experimentations on the implementation of novel inverse form-finding systems. Chapter 5 is an early experimentation in this direction for inverse form-finding of funicular structures towards a target shape. It demonstrates how an efficient and robust inner optimizer leads to more flexibility in the outer optimization loop. We also look at the importance of deriving derivatives of the objective function to obtain a good inverse form-finding framework.

Chapter 6 builds on and improves the simple inverse form-finding system for funicular surfaces of Chapter 5 to produce a feature-based representation of the design objective. Thanks to a functional representation of this objective, combined with automatic differentiation and a “perfect” forward simulation process – that is fast and always producing a result, we construct a workflow that not only does inverse form-finding towards a target shape, but to any objective that the designer can represent as a function of the properties of the current iteration. This is a novel way of looking at inverse form-finding design, that we find very promising in its potential uses.

Chapter 7 summarizes this dissertation, concludes on the main contributions and their potential impact, and outlines possible avenues for future work.

2 Background

This chapter critically reviews existing literature on computational design tools for construction systems with complex geometrical constraints, that let the designer express an intent to match.

2.1 Form-finding

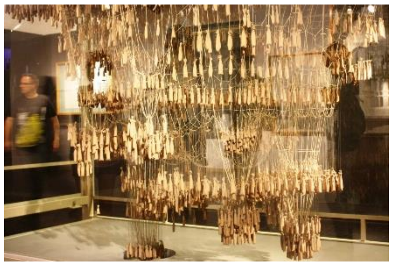
The concept of direct form-finding has been used since the advent of the construction of freeform shell and membrane structures in the 1970s and before, along with the development of computational structural analysis. One of the earliest references to the term is found in (Bubner, 1972) for the form-finding of pre-tensioned cable-nets. Before that, the term shape optimization was also used for structures like arch-dams (Deprez, 1968), and more generally for mechanical pieces. Adriaenssens et al. (2014) give a concise description:

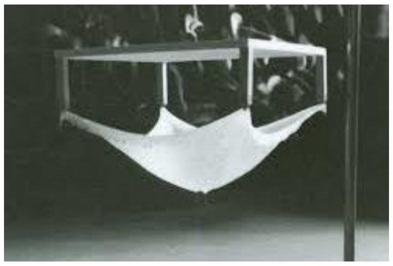
Form-finding is a forward process in which parameters are explicitly/directly controlled to find an "optimal" geometry of a structure which is in static equilibrium with a design loading.

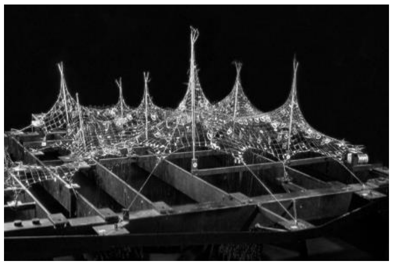
Typically, for shells made of heavy materials (concrete, masonry) or where creep is an issue (timber), this design load will typically be the dead weight; the parameters to be varied include the shape of the shell and its topology, and its boundary conditions. In this formulation, it is strictly a forward problem: there is no direct control possible on the resulting shape, only on the parameters generating it. Direct form-finding is the generation of a shape that efficiently solves a structural optimization problem.

2.1.1 Physical models

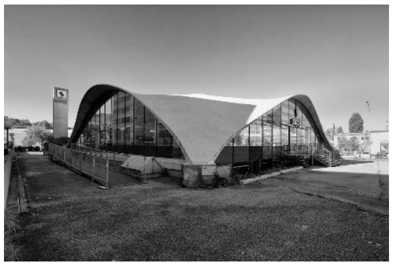
The earliest form-finding methods were probably physical ones. This is famously seen in the works and experiments of builders such as Eladio Dieste (1917-2000), Heinz Isler (1926-2009), or Frei Otto (1925-2015), and before them Antoni Gaudí (1852-1926), see Figure 2.1. They used scaled models to find funicular shapes for masonry structures, then membranes and concrete shells. The process is generally to hang a weighing chain or mesh and invert its shape to obtain a funicular shape under dead weight. Soap film models were also used for membranes. By adding weights, changing the boundary conditions and the topology, the resulting shape can be modified, However, this is a tedious process, as the model might lose tension in some parts for example, and many modifications will be time-consuming.


© chrispythoughts.wordpress.com


(Chilton et al., 2000, p. 37)


© FAR frohn&rojas

© Janna Goldsmith
Church of Colònia Güell,
Barcelona, Spain, 1914.
Arch. Antoni Gaudí


© jyhem @ www.flickr.com
Sicli building, Geneva,
Switzerland, 1969.
Arch. Heinz Isler.

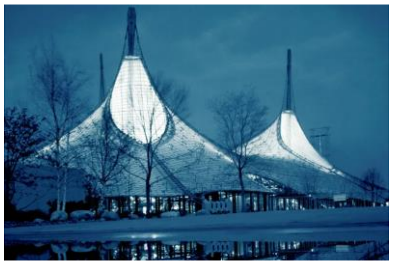

© Atelier Frei Otto Warmbronn
German Pavilion, Expo '67
Montreal, Canada.
Arch. Frei Otto

Figure 2.1: Physical form-finding models by Gaudí, Isler, and Otto (left to right), and the structures for which they generated viable shapes.

2.1.2 Computational form-finding

Form-finding solutions can also be found computationally, and a large literature of direct form-finding solvers exist. For shells, early methods were based on a simplification to a discrete mesh (Schek, 1974). For funicular shells, many of them reproduce the physical experiment of finding the equilibrium position of a hanging mesh. This is clearest for methods like the Particle Spring method (A. Kilian & Ochsendorf, 2005), that simulate a network of flexible springs attached to weighing nodes, and relax it until it stops moving. Although solving it in one direct matrix solve, methods like the force density method (Schek, 1974) solve for the same mechanical equilibrium of loaded nodes in a network of bars with axial forces. In fact, if the stiffnesses of the springs in this method are set to the force densities of the force density method, the form-found shape will be identical. Dynamic relaxation can also be used in this way, although in this case without the bar-and-node approximation and form-finding a free-hanging membrane (Brew & Lewis, 2007). Veenendaal and Block (2012) provide more details on the relationships between these methods for funicular and tension networks.

Computational form-finding is available for other construction systems – some are presented in Section 2.2. In many cases, dynamic relaxation is a possible choice, especially for the simulation of flexible materials, as its implementation tends to be more straightforward than other methods when starting from internal forces in the model. For example, it has been applied to tensile structures (Barnes, 1999), compression-only funicular structures (Bagrianski & Halpern, 2014), bending-active structures (Adriaenssens & Barnes, 2001), etc. Another method with a wide range of applications is the projective dynamics method (Bouaziz et al., 2014), which bridges finite element methods and position-based dynamics to create a form-finding tool that is simple to implement, yet robust and efficient. Applications include bending-active structures as well as pneumatics, cloths simulation, etc. Chapter 3 provides more details on these two methods and their inner workings.

For the designer, the interest resides in the fact that initial conditions and parameters of the simulation are much easier to change than in a physical model, allowing for a simpler exploration of form-found shapes. Structural typologies can also be combined and changed throughout the course of a design exploration, something that would have required tremendous effort with physical models. For example, dynamic relaxation and projective dynamics were both included in the very popular form-finding tool Kangaroo (Piker, 2016b, 2017a), directly integrated into the NURBS-based CAD system Rhino3d. Giving the designer a catalog of so-called “goals” to choose from to represent the internal forces in their constructive system, it opened up the methods of form-finding to a wider audience than before, and led to many complex applications combining multiple systems that traditionally would have each required a dedicated form-finding solution. However, the simulations can remain slow to run, and merely predict the shape of one instance of a structure. They do not guide the designer towards better structures that can be achieved with the chosen construction system, or provide intuition for finding structures that are closer to the design intent.

One method in particular here stands apart in this regard. Thrust network analysis (Block, 2009), which combines visualization from graphic statics and the force density method to create a form-finding tool that provides designers with both a form-found shape and an editable force diagram to act on the shape, gives them a little more control on the results. By redirecting the flow of forces to certain parts of the shape, the designer can create features like creases and ridges, see Figure 2.2. However, this requires a good understanding of the relationship between form and forces and is limited to funicular shape problems. Thrust network analysis is the basis of the successful masonry vaults design tool RhinoVAULT (Rippmann et al., 2012), and used for the best-fit optimization of funicular structures (Van Mele et al., 2014).

Systems with similar characteristics representing some information from the designer on the design intent include Combinatorial Equilibrium Modeling (Ohlbrock & Schwartz, 2015), where graph theory is used to control the qualitative behavior of funicular structures, and Algebraic 3D graphic statics (Hablicsek et al., 2019).

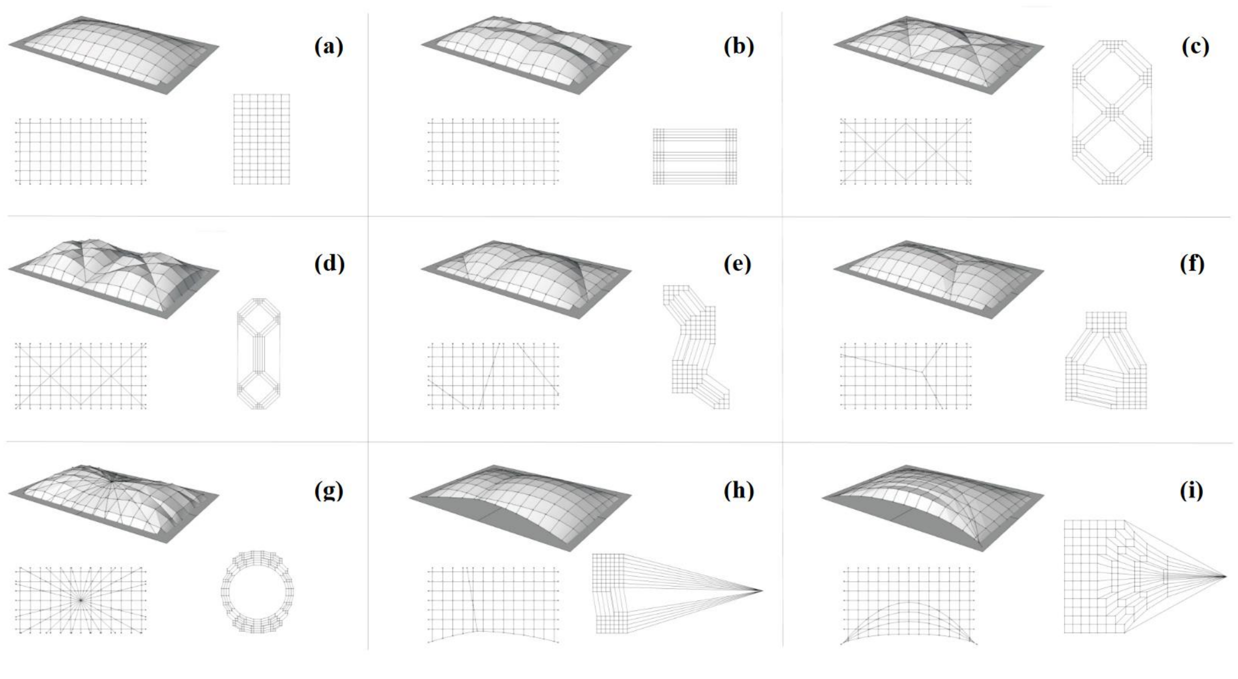


Figure 2.2: Various examples starting from a rectangular grid forming a funicular network, demonstrating how changing the force diagram can influence the resulting shape. From Block and Ochsendorf (2007).

2.1.3 Inverse form-finding

To overcome these issues, form-finding tools started incorporating other objectives in addition to solving for the mechanical equilibrium shape. Typically, this will require a more involved solving process, often based on mathematical optimization. That concept can be grouped under the loosely defined term "inverse form-finding", in reference to the term "inverse problem", which refers to the act of finding the causal factors that produced a set of given observations. (This is different from the concept of "inverse hanging models", where inverse refers to the inversion of gravity forces to generate compression-only shapes from tension-only models.) Here, we want to find the set of parameters and initial conditions to feed to the form-finding solver that will lead to the equilibrium shape that most closely matches the additional objectives.

Before looking at the different systems, it is important to notice that the vocabulary of inverse form-finding is not fixed in the literature. The term was introduced in 2010 in materials simulation, for example refereeing to finding the required initial shape of a steel blank to press-form it in a defined shape (Germain et al., 2010). This references a similar term, inverse deformations (Govindjee & Mihalic, 1998). On the other hand, papers especially coming from the Computer Graphics community tend to name form-finding processes a lot less, but use the same framework of iteratively modifying the inputs to an equilibrium shape predictor to get as close as possible to a target surface. Examples include (Panozzo et al., 2013; Skouras et al., 2014), they usually name the resulting tool a designing tool.

Form-finding systems with additional design goals can be subdivided into two categories: those that will solve the mechanical equilibrium exactly, that we call exact inverse form-finding methods or simply inverse form-finding methods, and those that do not that we call pseudo-inverse form-finding tools. True inverse form-finding tools solve the optimization problem ($P_{1.1}$):

$$(P_{1.1})\colon \min_{\boldsymbol{p}} f(\boldsymbol{x}), \text{ subject to } \boldsymbol{x} = \operatorname{argmin}_{\mathbf{y}} E_{\boldsymbol{p}}(\boldsymbol{y}). \tag{1.1}$$

Pseudo-inverse form-finding tools do not find a true mechanical equilibrium because they solve a relaxed version of ($P_{1.1}$), where both mechanical equilibrium and additional objectives are combined in a single objective function:

$$(P_{1.1b})\colon \min_{\boldsymbol{p},\boldsymbol{y}} f(\boldsymbol{x}) + E_p(y). \tag{2.1}$$

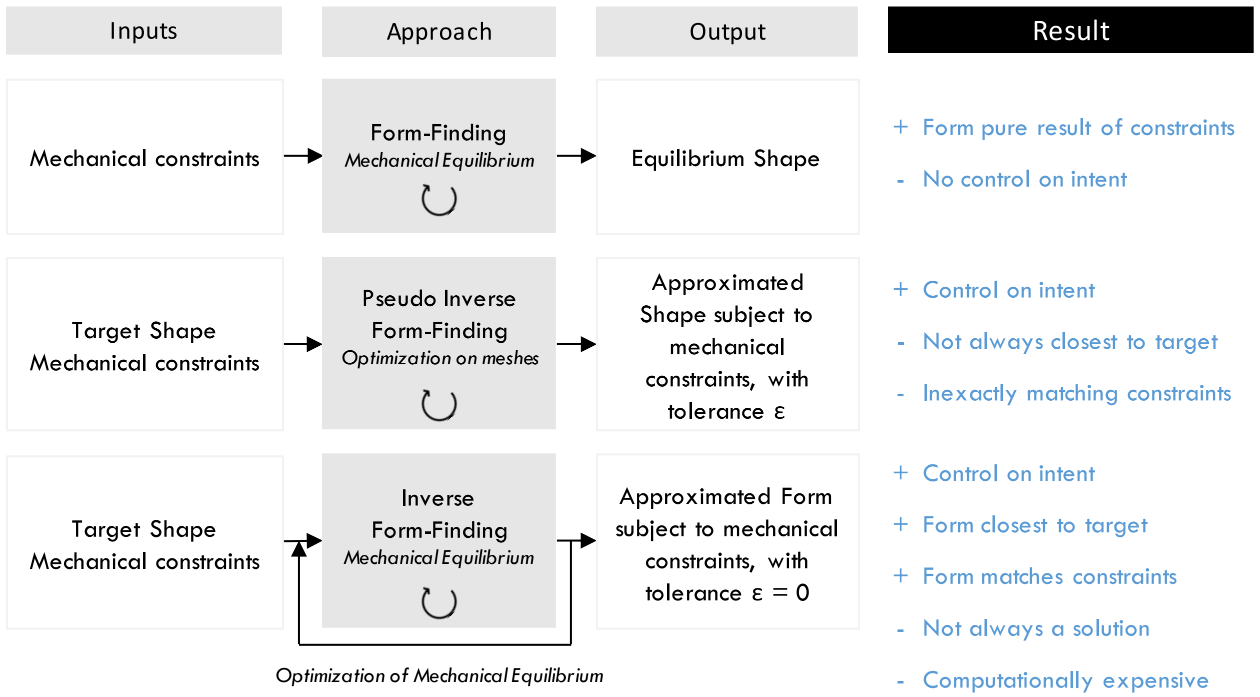


Figure 2.3: Classification of form-finding methods for design.

This simplifies the optimization process: while $(P_{1.1})$ will require two nested optimizers to solve, $(P_{1.1b})$ is usually solvable with existing non-linear optimization techniques. Pseudo-inverse form-finding typically discretizes the mechanical equilibrium problem to reduce it to an energy minimization problem on meshes. It then usually adds a term representing the distance from the form-found mesh to a target shape as an additional objective. By carefully selecting how these energies are weighed relative to each other, a shape can be found that is close to minimizing the mechanical energy and close to the target shape – if the mechanical system can realize such a shape. This is normally easier to solve computationally than the inverse form-finding problem, but cannot provide a guarantee on the quality of the mechanical equilibrium, unlike true inverse form-finding. Figure 2.3 summarizes their differences and relates them to direct form-finding. These concepts have been applied to a wide range of construction systems that are presented next, in Section 2.2.

In the form of Equation (2.1), pseudo-inverse form-finding can also be thought of as a multi-objective optimization problem, combining mechanical energy and the additional objectives representing the designer's intent in a single function. Multi-objective optimization for the early design of buildings has been studied for example by Brown (2019). When solving exactly the mechanical equilibrium problem of direct form-finding is not crucial, the techniques developed there are a very good alternative to true inverse form-finding and provide ways to efficiently explore the available design space.

2.2 Different construction systems and their constraints

This section of the literature review focuses on two types of geometrical constraints: constructive constraints that stem from the desired properties of mesh-like structures (rigid gridshells), and mechanics-based constraints that come from the equilibrium of a construction system. General approaches for arbitrary constraints are then reviewed, and some theoretical foundations are given. All the constraints considered are summed up in Table 2.1.

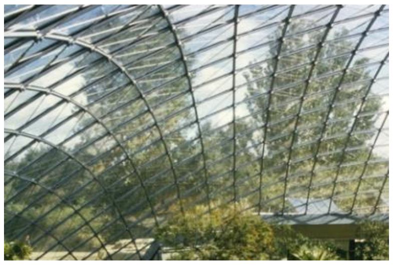
As discussed in Section 1.1.1, this dissertation focuses on creating tools for the inverse form-finding of two specific construction systems: bending-active structures and funicular shells. This section broadens the scope of possible construction systems, to study the literature on inverse form-finding methods for these systems.

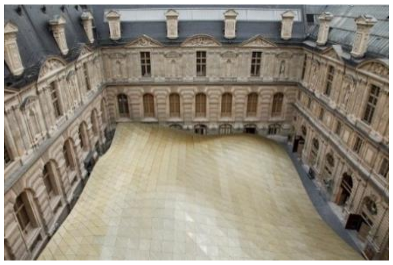
It is important to note that form-finding might also be used more as a shape generator than for finding shapes that are constructible with a structural system, so that sometimes these possible constraints are mostly used as an inspiration.

2.2.1 Constructive constraints

Mesh-like construction systems, such as the rigid gridshells of Figure 2.4 often need to accommodate geometrical constraints due to the relationships between nodes or edges introduced by the specific construction system. For example, in the Hippo house (Figure 2.4, left), the structure is covered by flat panels of glass, so the mesh has to be a mesh with planar quad faces. The covering of the Cour Visconti (Figure 2.4, center) similarly has flat quad panels, but for its substructure only, that describes a larger mesh supporting many faces. The Yas Viceroy hotel (Figure 2.4, right) has a façade made of continuous beams connected at pin-like nodes that align with the axis of the beams. This means the mesh needs to have a defined edge offset. Two other typical examples of constructability constraints are developable surfaces and piecewise developable surfaces. These occur for example when cladding a shape with metal sheets: each piece will be a developable surface. This is not a comprehensive list, each specific construction system giving rise to slightly different constraints, but it finds the most studied categories.

Table 2.1 (lines 1 to 3) sums up recent findings on the design of such surfaces. The direct form-finding column is not applicable here since form-finding needs a mechanics-based equilibrium principle to solve. Pseudo inverse form-finding problems have been solved in all areas, and inverse form-finding problems in all areas except offsetable meshes. Many of the papers in this area come from Professor Pottmann's research group at TU Wien, and even look at the interaction between these constraints.


© Karl Brösecke


© Musée du Louvre / A. Mongodin

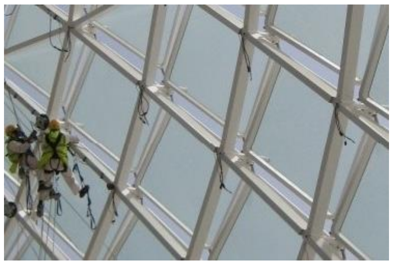

© Viceroy Hotels

Figure 2.4: (left) Hippo House, Berlin Zoo, 1996. J. Gribl arch., SBP eng. (center) Cour Visconti at the Louvre, Paris, 2012. Bellini & Ricciotti arch., HDA eng. (right) Yas Viceroy Hotel, Abu Dhabi, 2009. Asymptote Architecture, Front Inc. & Taw façade eng.

Table 2.1: Summary of recent developments in inverse form-finding, for various constraints.

System	**Direct form-finding**	**Pseudo inverse form-finding**	**Inverse form-finding**
Planar quads		(Eigensatz et al., 2010)	(Wallner & Pottmann, 2011)
Offsetable meshes		(Pottmann et al., 2007)	
Developable surfaces		(M. Kilian et al., 2008)	(Liu et al., 2006)
Funicular	(Schek, 1974)	(Vouga et al., 2012)	(Panozzo et al., 2013)
Bending	(Adriaenssens & Barnes, 2001)	(Quinn et al., 2016)	(Panetta et al., 2019)
Pneumatic	(Barnes, 1975)	(Sánchez et al., 2007)	(Skouras et al., 2014)
General frameworks	(Bouaziz et al., 2012)	(Tang et al., 2014)	

2.2.2 Mechanics-based constraints

Another large category of constraints is those for which a mechanical equilibrium has to be verified. These constraints led to the development of direct form-finding methods. Typical examples include funicular shells, where the shape of a shell is found such that the self-weight of the shell introduces only membrane forces, without bending, in itself. By eliminating the bending stresses of self-load, this leads to very thin shells, like on the Sicli building shell in Figure 2.5 (left). These shapes can be found, for example, by letting a weighted fabric hang from support points, then inverting the orientation. Active bending structures, like the Mannheim Multihalle elastic gridshell in Figure 2.5 (left) are made of a number of thin rods bent into shape. They produce easy-to-build shells, lightweight and resilient. Their shape is the result of the bending equilibrium of the rods. Lastly, pneumatic structures are membranes inflated to gain rigidity. They provide an extremely lightweight solution to large roofs, like on the Tokyo Dome stadium (Figure 2.5, right). Finding their equilibrium shape is especially difficult because of the potential for wrinkles in the membrane.

Historically, the structural design community produced several methods for the simulation of the large displacement processes involved in the construction of these structures, grouped under the direct form-finding term. The earliest applications date back to 1974, see Table 2.1. More recently, the computer graphics community has proposed design-oriented, inverse form-finding tools for these structures. They solve a large part of the grid in Table 2.1. Until recently, the inverse form-finding of active bending structures had received little attention from the computer graphics community, but a publication from July 2019 changed that by providing a fully integrated inverse form-finding design pipeline for a class of elastic gridshells that do not start from a perfectly flat regular grid (Panetta et al., 2019).

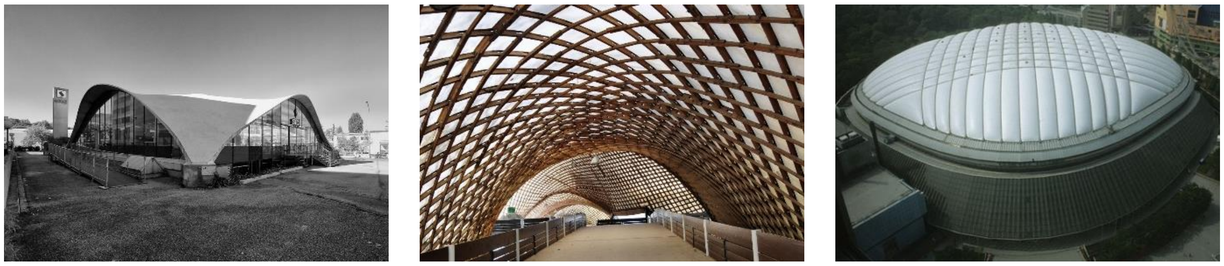

© jyhem @ www.flickr.com © Hubert Berberich © Yoshito Isono

Figure 2.5: (left) Sicli building, Geneva, Switzerland, 1969. Heinz Isler arch. (center) Mannheim Multihalle, 1975. F. Otto arch., Arup eng. (right) Tokyo Dome, 1988. Nikken Sekkei & Takenaka Corp., arch. & eng.

2.3 Design intent representation and formulation

The various form-finding tools also differ in how they represent the design intent. This intent can encompass many things; generally, in form-finding three categories exist for what might be controlled:

- The mechanical properties of the resulting shape, i.e. which kind of mechanical equilibrium does it describe.
- The materialization of the shape, including its possible discretization: topology, orientation of the mesh when it is discretized, sizing of the constitutive elements, etc.
- The visual and architectural properties of the shape.

In direct form-finding tools where the control on the final shape is necessarily limited, representing the design intent is usually limited to the first two categories. The designer can choose a solver like the force density method to generate shapes that will be funicular to a prescribed set of loads, for example. The materialization part is normally equally descriptive in direct form-finding, and the precise material properties of all the elements need to be prescribed in order to build the problem that the form-finder will solve.

Two systems stand out in this regard, that we have already mentioned. Thrust network analysis lets the designer generate axial stiffnesses for the elements of the funicular network by describing and modifying the flow of forces on the force diagram; this method was integrated into a design tool for funicular surfaces (Rippmann et al., 2012). Software like Kangaroo (Piker, 2016b) and ShapeUp (Bouaziz et al., 2012), because they can combine multiple mechanics-based properties to solve for, allow for finer control on this parameter. The reader is referred to Section 2.1.2 for a longer description of thrust network analysis and Kangaroo.

ShapeUp comes from Mark Pauly's group at the EPFL and is a general framework for the creation and deformation of constrained meshes. (Bouaziz et al., 2012) presents this framework and the solving algorithm. From an initial mesh, it iteratively finds small deformations that get the mesh closer to verifying a set of constraints while locally minimizing the deformation. The constraints are often a mechanical equilibrium but can also represent the distance to a target surface. This makes the algorithm a hybrid between direct and pseudo-inverse form-finding. Further developments give better guarantees on the minimizing properties, and introduce hard constraints that truly make the method a pseudo-inverse form-finding one (Tang et al., 2014). This flexibility is very desirable for more freedom in early design stages, when the final construction system might not be decided upon.

To control the visual and architectural properties of the form-found shape, the simplest and most common representation of the intent is by defining a target shape. In an inverse form-finding setup, approximating this shape will hopefully result in a reproduction of the features that the designer would like to obtain. This will potentially fail if one of the features cannot be built with the chosen mechanical system. Often, this lets the designer without a clue as to which part of the target or combination of features is impossible, so modifying the target will be difficult. One potential mitigation is to add weights describing which regions are more important to the designer, who will often "paint" these weights on an interactive design tool (Garg et al., 2014).

Another possible modification is to compare the form-found surface to the target shape using a more advanced metric than a simple point-to-point distance. For example, the designer could specify that only the curvature values of the target shape need to be matched, instead of the whole shape. The solver will then have more freedom to find a solution shape, but will still represent the important features of the target shape like creases, ridges or flat regions as these will have a characteristic curvature signature. Prescribing curvatures as a way of designing and editing surfaces has been proposed for example in Eigensatz et al. (2008), and direct form-finding tools exist to create surfaces of minimal curvature variation (a desirable property in industrial design) that match prescribed guide curves (Joshi, 2008). Design tools for optimal origami tessellations of shapes also include curvature as their main design objective (Dudte et al., 2016). Taking this idea further, it might be possible to represent intent and compute a difference to it using shape difference operators (Rustamov et al., 2013), as they represent how one mesh might be most efficiently and truthfully transformed into another, and define operators to transpose that deformation to a third mesh.

Finally, it is probable that to better represent the many subtleties of a design intent, it would be necessary to move from an inverse form-finding that is focused on matching one target shape, to one where a collection of small bits of information are combined and all optimized for. This is the same conceptual shift as what Kangaroo introduced for direct form-finding: instead of using separate direct form-finding systems for each mechanical constraint, combine them into a unified solution where the designer can pick and choose which constraints apply to which part of their system. This is the approach taken in Chapter 6.

2.4 Form-finding for fabrication

When generating shapes that will be directly fabricated using a specific construction method, design methods in the literature tend to take a point of view close to inverse form-finding. This can be because of a need to generate shapes that are results of direct form-finding processes, and respect an additional constraint that is simple to represent in the form-finding process. For example, John Orr (2012) designs fabric formworks for concrete beams and shows how each cross-section is found using a direct form-finding process for the shape of an elastica curve (a bending-active structure made of a single rod). This is adapted to accommodate the fact that successive sections need to have a progressively varying shape. Similarly, Dessi-Olive (2017) uses form-found elastica curves as guides for a masonry construction system. By manually selecting elastica shapes that are close to the shape decided upon for the masonry system, more efficient structures are built. This method was also used for the construction of a tile-vaulted shell (Block et al., 2016).

A more complete inverse form-finding method is described by Veenendaal (2017) as “constrained form-finding”, where viable shapes are found for flexible formworks for concrete shells. For example, the form-found shape of a pre-tensioned cable network, loaded with the weight of fresh concrete, is optimized so that it is close to describing a funicular shape under the concrete weight. Designing fabric formwork for concrete objects and structures was also studied by Zhang et al. (2019), who built an inverse design tool that solves for many of the constraints imposed on such formworks, like fabric paneling, wrinkling and casting deformations. One drawback of that method is its speed, generating a design in several hours.

2.5 Implementations of designer-guided form-finding systems

For these systems where the designer has control over the form-found shape, three broad categories of implementations can be defined. The first one concerns systems where the result of the direct form-finding can be found constructively from the initial conditions, without needing to resort to an optimization loop. This is found for example in Marionette meshes (Mesnil et al., 2016), which defines a class of meshes that define only planar facets. The full design space of meshes with planar facets is not generated by this method, but instead it finds a unique solution from two boundary curves in elevation and a projection of the mesh on the horizontal plane. It does so by systematically applying a simple linear equation to each facet of the mesh, which is both fast and robust. By simplifying the design space in this way, the designer is provided with a form-finding tool that has perfect control over the final geometry. This type of tool is regularly applied for a "geometrically-constrained design strategy" (Bagneris et al., 2008), and has been extensively used in architectural shape generation, for example on ruled surfaces for easy framework construction of concrete shells by Candela (del Blanco García & García Ríos, 2019). Implementation for these systems tends to be very focused on geometry, and thus dependent on the CAD framework used, and is generally similar to an implementation of a script for parametric design.

The second category of implementations deals with the pseudo inverse form-finding systems. There, the focus is normally on the optimizer used, and the computation of gradients of the energy being minimized. The solvers can be general-purpose ones from commercial packages (Jacobson et al., 2011; Jiang et al., 2017), or open-source implementations (Skouras et al., 2014). Some systems define custom-made optimization routines (Vouga et al., 2012). Computation of the gradients involves careful mathematical definitions of the energy being minimized, generally on a mesh so that techniques from discrete differential geometry (see Section 2.6) can be utilized, and great care is taken on developing expressions that lead to efficient numerical computations. Most of these systems then use a geometry representation based on meshes, with arrays storing the positions of the vertices and their connectivity. This leads to very optimized and efficient implementation of the optimization problem, albeit being hard to extend once finished.

Lastly, systems like Kangaroo (Piker, 2016b) and ShapeUp (Bouaziz et al., 2012) where many constraints can be combined usually implement one simple solving method, and apply it to an energy that is only defined at runtime, once the designer has built their objectives. The energy computation is typically a loop over the collection of objects sent to the optimizer, each object defining the method to use to compute its contribution to the energy. This is much less efficient to compute than the vectorized operations of the previous category, but has the advantage of greater design flexibility.

2.6 Different academic communities

Beyond the structural design community, two other academic communities are also interested in the optimization of geometrical objects, and generating shapes that match prescribed constraints. We have already mentioned the computer graphics viewpoint, which typically produces pseudo-inverse form-finding tools focusing on the design of one system. A second group is (discrete) differential geometry, and its application in architectural geometry.

There is a limited precedent of literature on rigorous modern geometry for architecture. One well-known book filling that void is the result of the research group led by Helmut Pottman at the University of Wien (Pottmann et al., 2015). A large part of the book is dedicated to presenting classical shapes of descriptive geometry, their properties, and interactions. It then details the mathematical background for NURBS curves and surfaces used in modern freeform CAD software. These two sections suffer from the limitations mentioned in the introduction: they can only represent the constructability constraints in very specific cases, or even not at all until they are rationalized. The last section is dedicated to recent research by the authors in constrained discrete surface generation and rationalization. It presents several typical requirements of architectural shapes and proposes concepts that solve those requirements. It demonstrates the large improvements that discrete differential geometry can bring to architectural shape representation, and emphasizes the need for constructive and iterative algorithms – this is not always considered by differential geometry theorists. However, the authors focus mainly on planar quadrilateral meshes as a construction method, and never consider the structural performance of the designs. Lastly, this book is destined to architects and designers and lacks most of the mathematical derivations.

For continuous differential geometry, most of the introductory literature comes from physics – many of the concepts of differential geometry were developed for solving the equations of general relativity. A reference textbook in the domain is (Frankel, 2011). It presents all the mathematical theory needed for the study of manifolds, with a focus on 2, 3, and 4-dimensional objects. However, discrete objects are not studied. Current research in discrete differential geometry in low-dimensional spaces is led by computer graphics groups. Crane (2014) gives a course in this domain with a focus on efficient iterative algorithms for smoothing and optimization of meshes.

Discrete differential geometry approximates continuous differential geometry on meshes and other simplicial surfaces. Because architectural surfaces are often realized with discrete elements, meshes seem like a good representation of the built reality. But their constitutive laws do not reproduce continuous reality, they truly describe a discrete object, and to approximate the behavior of a curved surface new mathematical properties can be attached to the elements of meshes. This can lead to robust and efficient formulations of energies defining mechanical equilibriums, for example for bending-active structures the model of Bergou et al. (2008) was shown to be very powerful, see Chapter 3.

2.7 Summary

Direct form-finding tools allow designers to generate shapes matching prescribed constraints like a mechanical equilibrium. Because the results of these methods tend to be very hard to control, inverse form-finding was developed to allow the steering of form-found shapes towards additional objectives like a target surface. Such tools were initially taking on a relaxed version of the optimization problem defined by inverse form-finding, leading to pseudo-inverse form-finding where the designer has control over the form-found shape, but it will not match exactly the mechanical equilibrium constraints. True inverse form-finding on the contrary requires nested optimization loops, a setup that is very demanding on computational power and where the inner optimizer – a direct form-finding solver, or forward simulation – is put under repeated and automated use on a wide variety of inputs. This means that this solver needs special attention in terms of its speed, reliability and accuracy.

True and pseudo-inverse form-finding are generally focused on one single shape target objective, and similarly direct form-finding solver used to be limited to one single mechanical equilibrium constraint. Newer systems let the designer specify varied constraints in one unique framework for direct form-finding, and combine them to produce new shape generators that were not considered before.

Two mechanical equilibriums of interest in this dissertation are bending-active structures and funicular structures. Until very recently, true inverse form-finding had not been applied to any bending-active structures; current solutions apply to a subset of them. Additionally, implementing one such system puts very specific requirements on the forward simulation. By carefully studying these requirements on various methods for the direct form-finding of bending-active structures (Chapter 3), we will be able to improve their implementation and gain a better understanding of how they work and perform (Chapter 4). We also look at finding good representations of the variables (initial conditions and parameters) sent by the outer optimizer to the inner one on different case studies.

Funicular structures have seen more interest in terms of the inverse form-finding tools developed for them, but will prove to be a good model system for experimentations on the implementation of novel inverse form-finding systems (Chapter 5). Specifically, by implementing an inverse form-finding tool in an automatic differentiation framework, we can imagine a system that is not focused on a single target shape objective (Chapter 6).

3 Improving the speed, accuracy and reliability of form-finding processes for bending-active structures

According to shell expert Chris Williams, form-*active* structures are a broad category of structures that react to external loads and constraints with large deformations, in contrast with more common form-*passive* structures such as frames or rigid shells (Williams, 2014). (This classification is different from Engel's (1967) classification where form-active structures are contrasted with section-, surface- and vector-active structures.) These structures are lightweight, since they do not resist forces by direct material rigidity but by geometric deformations, and resilient because they can flex instead of breaking.

However, predicting the rest shape of these structures and their behavior under different loading conditions, is challenging because of this deformability, involving non-linear processes. This rest shape needs to be found using direct form-finding tools. In this chapter, we aim at improving the quality of such form-finding tools for one common subset of form-active structures: bending-active structures. Fast and reliable form-finding tools will be of great importance for Chapter 4, where they will be used repeatedly in an optimization loop – without any real-time control possible by the designer on the form-finding process – to find bending-active structures that are closest to an objective shape.

3.1 Introduction

3.1.1 Problem statement

Bending-active structures are structures where structural elements are linear rod elements or elongated plates that deform in bending, often combined with fabric membranes to create lightweight structures. The construction process of these structures is fast and accommodating to large tolerances, making them a great solution to temporary covering problems, and to problems where adaptivity and reconfigurability are required (Coar, 2010). They need not be temporary structures, some examples are shown in Figure 3.1, like non-regular rod assemblies (Coar, 2012), regular grids of rods (Ban, 2003), and hybrid structures (Cuvilliers et al., 2017).

The computational design process of a bending-active structure always involves finding the equilibrium shape of the structure. It is crucial that this final shape is predicted accurately, which remains a challenge with available software tools. For example, the final shape could be the support of another element, so that errors in the shape would result in incompatibilities with the secondary elements (Olcayto, 2007). Additionally, small errors in shape can lead to larger errors in the internal stresses predicted for the structure (Douthe et al., 2010), and even larger errors in the prediction of non-linear processes such as buckling (Mesnil et al., 2015).

In the simulation process, the topology and connection types of the structure are first defined, along with the length of the bars and the boundary conditions such as anchors in the ground. An iterative algorithm (the specific types available are described in Section 3.2.5) then relaxes the rods' positions until they are at equilibrium.

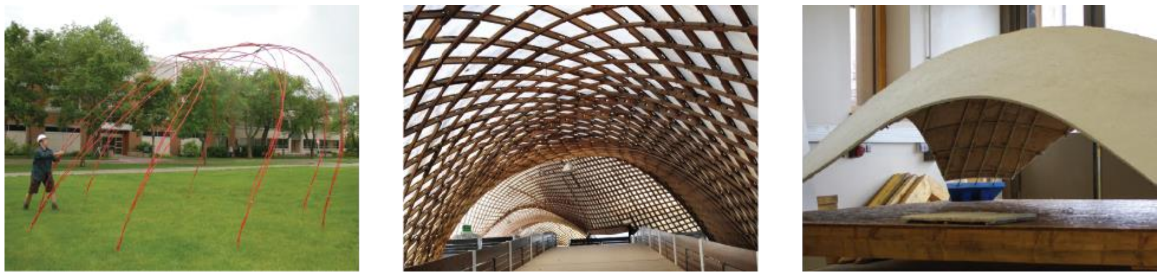


Figure 3.1: Three examples of bending active structures. From left to right: bending-active frame, bending-active gridshell with a flexible membrane, and hybrid bending-active gridshell integrating a rigid shell. Central picture © Hubert Berberich (CC-BY 3.0).

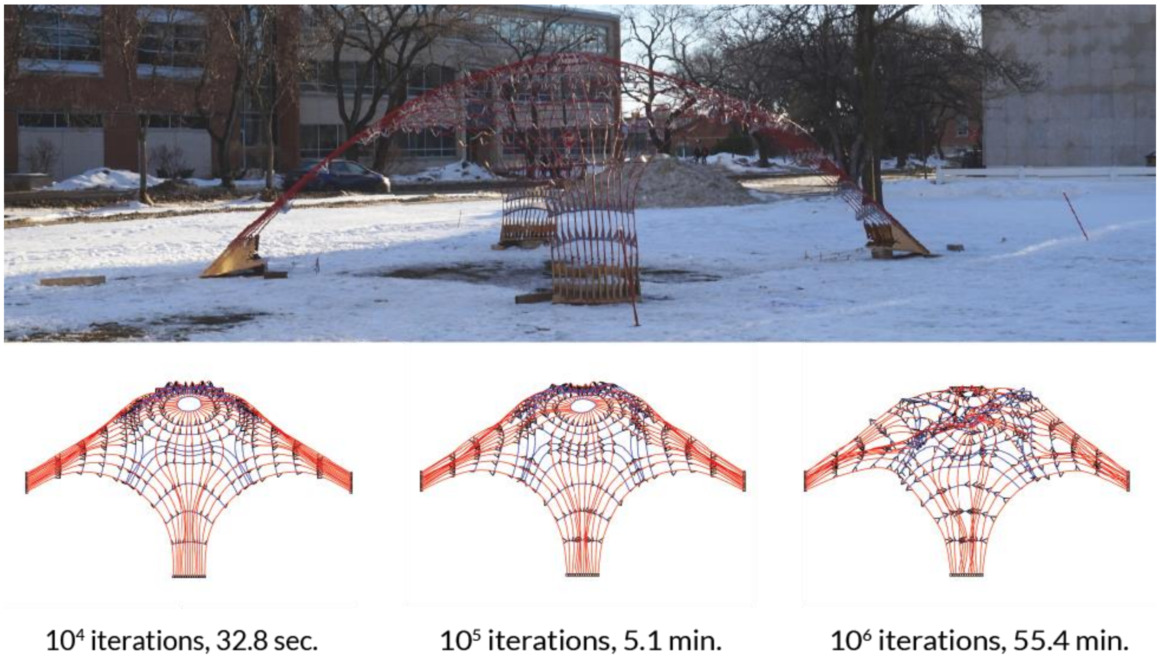


Figure 3.2: Example of a computational reproduction of the construction process for a bending-active structure (Coar et al., 2017). The runtimes are extracted from one run of the Kangaroo 2 solver, a commonly used architectural design tool, on the structure.

However, this iterative algorithm is at risk of being too slow for interactive explorative design, unreliable (giving unexpected results) and inaccurate (giving out-of-equilibrium results). It has no guarantee of convergence, and can often produce a false sense of accuracy and definiteness. An example of a typical failed design process is represented in Figure 3.2. An attempt is made to reproduce the shape of a complex bending-active structure made of a two-layer grid. When a solver is run on this problem, it initially seems like it finds an equilibrium. However, running the solver for more iterations–and a longer time–shows that this is not the case. It could be that the solver parameters were not appropriate, or that the simulation did not accurately represent the construction process. In any case, the first shapes obtained after smaller run times are not to be trusted, as the nodal positions and forces they give are more representative of the initial configuration of the simulation than reality. This raises awareness to the fact that convergence settings need to be carefully selected to produce reliable results. This problem is detailed in Section 3.5.4.

Several algorithms and frameworks have tackled this problem, discussed in Section 3.2. However, these algorithms have not yet been comparatively evaluated specifically for the modeling of bending-active structures. As a result, building a design tool for one structure requires a lot of trial and error across the different possibilities. In addition, the stopping criterion – determining for how long the solver will try to improve its solution – is critical to the quality of the results, yet its setting is often overlooked. In this chapter, different algorithms are compared, both from existing available tools using:

- dynamic relaxation (Brew & Brotton, 1971) in Kangaroo 1 (Piker, 2016b), and
- a variant of the projective dynamics method (Bouaziz et al., 2012) in Kangaroo 2 (Piker, 2016b),

and from general-purpose optimizers:

- a custom implementation of the dynamic relaxation method,
- augmented Lagrangian (Birgin & Martínez, 2008) with L-BFGS (Byrd et al., 1995), and
- SLSQP (Kraft, 1988).

The quality and choices of stopping criteria are compared on three benchmark metrics: speed, reliability and accuracy. Guidelines for the parameters to use are given.

3.1.2 Comparing form-finding processes to general-purpose optimizers

It might seem unusual at first to compare dynamic relaxation and projective dynamics against general-purpose optimizers. The former two represent the problem to solve as evolving a (pseudo-)physical system, made of nodes and elements on which internal and external forces are applied and moving the nodes in the direction of forces until an equilibrium is reached thanks to a damping mechanism; while the latter generally minimize the global energy of the system.

However, both dynamic relaxation and projective dynamics are equivalent to an energy minimization process. For projective dynamics, this is shown in Narain et al. (2016), showing equivalency to the minimization of the sum of the potential energies associated with the forces defined on the system, using the alternative direction of multipliers method. Similarly, dynamic relaxation can be seen as a type of accelerated gradient descent method (Nesterov, 1983) on the elastic energy of the system. Minimizing the elastic energy of a physical system using that method implies moving the nodes (the variables in the optimization process) in the direction of the gradient of the energy (the "gradient descent"), that is in the direction of the forces applied at each node. This direction is "accelerated" by adding to it a function of the movement at the previous iteration; this is exactly like tracking the velocity of each node and carrying it forward in time through some inertia as is done in dynamic relaxation. The damping part of dynamic relaxation is reproduced in the function giving how much previous displacements influence the current iteration.

Then, the structures created in the bending-active process are in fact minimizing the bending energy of the rods, under the constraint of inextensibility of the rods—the bending stiffness is typically orders of magnitude lower than the axial stiffness for an elongated rod. Alternatively, the inextensibility constraint can be relaxed by calculating the energy of the compression of the rod, the energy to be minimized is then the sum of the bending and stretching energies:

$$(P_{3.1}):\ \min_{x} \sum_{i=0}^{n} \frac{EI(\kappa \boldsymbol{b}_i)^2}{\bar{l}_i}, \text{ s.t. } \forall i, l_i = \bar{l}_i, \text{ or :} \tag{3.1}$$

$$(P_{3.2}):\ \min_{x} \sum_{i=0}^{n} \frac{EI(\kappa \boldsymbol{b}_i)^2}{\bar{l}_i} + ES\left(l_i - \bar{l}_i\right)^2 . \tag{3.2}$$

where $\kappa \boldsymbol{b_i}$ is the curvature binormal vector at node i, that rotates the edge before node i into the edge after it, and $\bar{l}_i$ the length of the edges coming to node i. The bar designates initial values, that are kept constant for edge lengths. How different implementations and discretizations compute these values from the nodal positions x is explained later in the chapter.

3.1.3 Organization of the chapter

Dynamic relaxation has been used very often for bending-active simulations since its creation; in this chapter we investigate whether other methods might be useful. This method gives intermediate steps that are physically meaningful. However, in a design problem we only care about the end result – the static equilibrium position – so that there might be more efficient methods available where the intermediate steps taken are not meaningful. This is why we chose to compare against general-purpose optimizers that do not give physically meaningful intermediate steps.

In each case, the overarching goal is to find a solver for the direct form-finding problem of bending-active structures, that can be run inside an outer optimization for inverse form-finding problems. This leads to the three criteria for comparing the solvers, speed, accuracy and reliability. Additionally, because of the limited control exits on the inner solver in an inverse form-finding process, the various parameters used in each solver need to be simple to decide and fix throughout the simulation. The objective is to find one algorithm that will be fast, accurate and reliable on a wide range of inputs, for a given and fixed set of configuration parameters.

The software tools selected are widely used in the architectural design community and implement methods that have become standards for form-finding simulations (Section 3.2). By comparing their predictions against the analytical result for the planar elastica (Section 3.3), we find guidelines for the simulation setup that produce accurate results without compromising too much on execution time (Sections 3.5.1 and 3.5.2). Finally, we use these guidelines for the simulation of a larger structure and show that they lead to reliable results (Section 3.5.4).

3.2 Literature review: precedents in simulation of bending-active structures

3.2.1 Simulation of elasticas

At the heart of every bending-active structure is the elastica, the mechanical equilibrium problem describing the shape of one elastic rod spanning between two supports (Levien, 2008). Simulating one elastica is a complex problem, for three reasons:

- It is a highly geometrically nonlinear problem; small changes in the boundary conditions leading to large changes in the shape and different stability regimes can coexist (Goyal et al., 2008);
- The interactions between bending and torsion are complex, and simulating them involves keeping track of more than three degrees of freedom at each discretization point (du Peloux et al., 2015);
- Bending and axial compression in the rod operate at very different stiffnesses, leading to ill-conditioned numerical problems when they are modeled simultaneously. Considering non-extensible rods requires more advanced constrained optimization algorithms (Bergou et al., 2008).

3.2.2 Simulation of complex structures

For active-bending structures, elasticas and other elements are then assembled to form a complete structure. The loads are usually light live loads of wind and impact. This results in very flexible structures. The assembly and erection processes in particular always incorporate large displacements that conventional finite element method packages for structural engineering such as Robot or RISA-3D cannot easily represent (although more powerful generalist packages can be used, as described below). As such, these new structural typologies and systems require new approaches for engineering.

The approach we consider here has the formulation of a form-finding problem: given the definition of a bending-active structure, with its rods, connections, boundary conditions and loads, what is its equilibrium shape? Two solving methods are typically used in this approach:

- Generalist finite element method packages such as Abaqus are available for bending-active structures (Nabaei et al., 2013). The resulting models are often accurate and easily calibrated with physical quantities; however, they tend to be slow by default, and do not reliably find the main equilibrium. They are also poorly integrated with the usual architectural design tools.
- Discretized elements in bending have recently attracted considerable attention from the computer graphics community, mainly for the simulation of dynamic systems such as hair (Nealen et al., 2006). These methods tend to be faster and give predictable results for dynamic problems, but are

sometimes not accurate enough for the simulation of a static problem, and require great care in tuning (Bergou et al., 2010). Derivatives of these tools have been integrated in architectural design tools, most notably the Shape-Up library (Bouaziz et al., 2012). It is not always clear how each algorithm can be calibrated to give results in meaningful physical units (Anders et al., 2016).

Specifically for elastic gridshells, Sakai et al. (2020) provide a new beam bending energy discretization, using the intersecting rods to keep track of surface normal and beam torsion. The authors implement this method in two direct form-finding algorithms, dynamic relaxation and one general-purpose non-linear optimizer (SNOPT), comparing results and number of iterations for each. This is close to the objectives of this chapter; however, limited in applicability to elastic gridshells rather than all bending-active structures, and does not provide information on the relative speeds of each method, only the number of iterations they need to reach convergence.

3.2.3 Computational design of bending-active structures

Examples abound for bending-active structures where a digital prototyping tool was critical to the design process. The CITA group and the Complex Modeling project produced several towers made from fiberglass rods in a water-drop shape, stacked and tensed by a tailored-designed membrane (Tamke et al., 2016), and elastic gridshells (Nicholas, 2013). They used specialist tools built on top of Kangaroo (described below) for the design. The ITKE created several examples such as Flectofin®, a large-size flapping mechanism (Lienhard et al., 2011), and umbrella-shaped bending-active structures (Lienhard & Knippers, 2015), using custom-made non-linear finite element method procedures. Several recent elastic gridshells also provide interesting examples using design tools based on the dynamic relaxation method (described below) (Douthe et al., 2010; Mork et al., 2016). Each time, the authors show how only a specialist use of form-finding tools made a more comprehensive design process possible. Additionally, all of the examples cited in this section had to incorporate tolerance-correction systems to overcome the accuracy shortcomings of the software. This shows a strong need by designers for accessible tools that can simulate bending-active structures.

3.2.4 Design tool reviews

Reviews of custom-made tools and frameworks for bending-active structures exist. For structures made of membranes and elasticas, Van Mele et.al. (2013) and Ahlquist and Menges (2013) look at the influence of the quality of the simulation tool on the design process, and improve on its speed or ease of use. However, they do not closely consider reliability or accuracy. More comprehensive reviews of modeling and design techniques are also available (Lienhard, 2014; Lienhard et al., 2013), detailing how design methods stemming from different generations of design tools created new categories of bending-active structures. These present very detailed analysis of construction systems, but do not focus on verifying the accuracy of simulation tools for a range of conditions. A comparison of simulation results to analytical results, is found in (Adriaenssens & Barnes, 2001); however, the simulations are based on the dynamic relaxation method only and no additional methods are considered.

Although these cited studies manage to simulate simple bending-active behavior, they do not address a comparison between simulated and built models. This makes the accurate fabrication of physical elements from simulated forms an unreliable proposition, showing a need for adequately verified and accessible software for the direct form-finding of bending-active structures.

3.2.5 Bending-active algorithms

There are two commonly used form-finding tools for simulating bending-active structures considered in this chapter: dynamic relaxation and projective constraint-based solving, respectively implemented in the Kangaroo 1 and Kangaroo 2 software packages (for Rhinoceros3D / Grasshopper). These tools are tested in Section 3.3. They are widely used in the architectural design community, with a combined number of downloads of close to 250,000 at present (Piker, 2016b), and free. Additionally, they both share the same author-developer and programing language (C#), allowing for a comparison of the algorithms, not only their software implementation.

Additionally, this dissertation presents a custom implementation of the dynamic relaxation method (Brew & Brotton, 1971), with kinetic damping (Barnes, 1988) and fast manifold projection (Bergou et al., 2008) for constraints enforcement, and compares it to two generalist optimization methods: SLSQP (Kraft, 1988) available from the scientific computation library Scipy; and L-BFGS (Byrd et al., 1995) in an augmented Lagrangian (Birgin & Martínez, 2008) scheme for constraints enforcement, from the optimization package NLOpt (Johnson, n.d.).

3.2.5.1 Kangaroo 1

Kangaroo 1 (Piker, 2016b) (this work uses version 0.099) implements a dynamic relaxation solver. Dynamic relaxation is a time discretization of the dynamical behavior of physical systems (Brew & Brotton, 1971), introduced in the 1960s. The general idea is to simulate the dynamic behavior of the structure with carefully chosen damping parameters to reach an equilibrium state as quickly as possible. As such, it is easily related to the physical parameters of the rods, but introduces new parameters that have no influence on the final result but can lead to instabilities: mass and damping. At each time step of length h, the algorithm integrates the second law of dynamics using a sympleptic Euler scheme:

$$\begin{cases} \boldsymbol{M}\ddot{\boldsymbol{x}}^{(n+1)} = \boldsymbol{M}\dot{\boldsymbol{x}}^{(n)} - h\dfrac{\mathrm{d}E_{bending}[\boldsymbol{x}]}{\mathrm{d}\boldsymbol{x}} \\ \boldsymbol{x}^{(n+1)} = \boldsymbol{x}^{(n)} + h\dot{\boldsymbol{x}}^{(n+1)} \end{cases}, \tag{3.3}$$

where $\boldsymbol{M}$ is the mass matrix and h the time step.

Specifically, Kangaroo 1 implements a time integration of Newton's second law using a semi-implicit Euler method (Senatore & Piker, 2014). At every iteration with time step Δt after time t, the solver computes the nodal velocities $v_i^{t+\Delta t}$ using nodal forces F_i^t from the previous update and masses M_i, then gets positions $x_i^{t+\Delta t}$ using the new velocities:

$$\begin{cases} v_i^{t+\Delta t} = v_i^t + \Delta t\dfrac{F_i^t}{M_i} \\ x_i^{t+\Delta t} = x_i^t + \Delta t v_i^{t+\Delta t} \end{cases}. \tag{3.4}$$

This is a conditionally stable integration scheme with a wide stability region, and that is sympleptic meaning it has very good energy conservation (Hairer et al., 2006). Dynamic relaxation has to add virtual unit masses on the points that do not have one defined so that their dynamic behavior is defined. The time step is chosen to fall in the stability region of the method, given the nodal masses (Senatore & Piker, 2014). The damping of the system is done here with the kinetic damping scheme, resetting the system's velocities every time the kinetic energy reaches a peak (Barnes, 1988). Dynamic relaxation has been used for a wide range of form-active structures (Bagrianski & Halpern, 2014; Barnes, 1999), including active bending structures (Barnes et al., 2013; Liew et al., 2016). These papers also use kinetic damping, and use a central-difference time integration scheme that leads to the same update—and properties—as the semi-implicit Euler method albeit considering speeds at a different time:

$$\begin{cases} v_i^{t+\Delta t/2} = v_i^{t-\Delta t/2} + \Delta t\dfrac{F_i^t}{M_i} \\ x_i^{t+\Delta t} = x_i^t + \Delta t v_i^{t+\Delta t/2} \end{cases}. \tag{3.5}$$

This makes Kangaroo 1 a representative tool for contemporary implementations of the dynamic relaxation method in bending-active simulations.

In order to converge to a stable position, a damping mechanism is introduced, we selected the kinetic damping available in Kangaroo 1. The kinetic damping scheme is one of the most stable available for dynamic relaxation. It simply resets the velocities $\dot{\boldsymbol{x}}_i$ to 0 every time the kinetic energy reaches a peak. Generally, this algorithm can be thought of as an equivalent of accelerated gradient descent (Nesterov, 1983), with less carefully crafted acceleration but easy physical interpretation.

3.2.5.2 Kangaroo 2

Kangaroo 2 (Piker, 2016b) (this work uses version 2.1.2), uses projective constraint-based solving, a method developed in the computer graphics community and made available to wider audiences through the general-purpose simulation tool Shape-Up (Bouaziz et al., 2012). At each iteration, the solver moves closer to the equilibrium position by projecting the positions on the sets of constraints representing the relationships between them. The solution is a physical equilibrium with correct derived forces if the constraints are physically accurate. The solving process used by Kangaroo 2 is a specialized version of the Shape-Up algorithm: it uses the same projective based constraints and accelerates movements using a virtual velocity attached to each vertex. This code is not entirely available to the public, slightly obscuring this process. However, simple experiments – observing the movement of a single particle in a singular force field – and explanations by the developer (Piker, 2017b) give some insight. They show that the algorithm resembles the kinetic damping of dynamic relaxation (Barnes, 1988), resetting the vertices' velocities when the forces change direction, but using forces derived from the projective dynamics method (Piker, 2016a).

3.2.5.3 Custom dynamic relaxation

In our custom implementation of dynamic relaxation, we reproduce the same time integration and damping mechanisms as in Kangaroo 1, but added a mechanism for solving the lengths constraints, instead of adding them a stiff energy term.

At each time step, it is possible to strictly enforce non-linear constraints such as the lengths constraints by projecting the positions onto the closest point on the constraint manifold, correcting the velocities by the appropriate amount. This is in accordance with constrained dynamics, so it does not change the physical properties of the algorithm. However, finding the closest point on the constraint manifold can be time-consuming; an approximation used in Bergou et al. (2008) and Goldenthal et al. (2007) is the fast manifold projection method. Effectively, it finds a close-by point exactly on the constraint manifold by taking a succession of as small as possible steps, changing the bending energy to the second order in the time step. The method repeats the following iteration until convergence is achieved:

$$\begin{cases} \text{Solve } h^2(\nabla \boldsymbol{C}[\boldsymbol{x}^*]\boldsymbol{M}^{-1}\nabla \boldsymbol{C}[\boldsymbol{x}^*]^T)\delta\boldsymbol{\lambda} = \boldsymbol{C} \\ \delta\boldsymbol{x}^* = -h^2\boldsymbol{M}^{-1}\nabla \boldsymbol{C}[\boldsymbol{x}^*]^T\delta\lambda \\ \boldsymbol{x}^* \leftarrow \boldsymbol{x}^* + \delta\boldsymbol{x}^* \end{cases}, \tag{3.6}$$

where $\boldsymbol{C}[\boldsymbol{x}^*]$ is the constraints vector and $\nabla \boldsymbol{C}[\boldsymbol{x}^*]$ its gradient. This is the method implemented in this chapter.

3.2.5.4 General-purpose optimizers

L-BFGS and SLSQP are two other methods of interest as they are very established generalist optimization methods, despite seldom being used in bending-active research. L-BFGS (Byrd et al., 1995) is a quasi-Newton solver. It cannot accommodate strict non-linear constraints directly; instead, it has to be embedded in an augmented Lagrangian method (Birgin & Martínez, 2008), this is provided automatically in NLOpt (Johnson, n.d.). SLSQP (Kraft, 1988) uses sequential quadratic approximations of the problem to obtain an optimum. It can solve subject to arbitrary non-linear constraints, as these are directly passed to the quadratic problem solve. Each solution of quadratic approximations gives a direction for a Newton-like line-search in the complete problem. SLSQP has been used for the optimal design of flexible actuated structures, although not for the form-finding of the beam (Maraniello & Palacios, 2016).

3.3 Methodology: Single elastica comparison

Before studying the simulation of complex bending-active structures containing many members, we first focus on simulations of simple, single-curve elastica problems, as they represent the elementary problem of all bending-active structures and can be solved exactly for forces and geometry using analytical equations. This allows for a direct comparison between the algorithms' and analytical results. All algorithms for active bending can represent the elastica, and the quality of their results on multiple-rods structures depends directly on their results for one elastica.

For each solver, we look for parameter settings that will give a predictable accuracy in the shortest possible time. For this, we run the solver with varying parameters on a range of boundary conditions for the elastica. For each set of parameters, the goal is to have consistent accuracy and speed across the range of boundary conditions: this represents the different shapes that the designer will encounter when modeling a set of elasticas.

3.3.1 Elastica problem

We consider a beam of length L, with Young modulus E, section A and moment of inertia I, as described in Figure 3.3, top. Analytically, the elastica is the solution of the moment equilibrium in the beam (Audoly & Pomeau, 2010):

$$EI\frac{d^2\theta}{ds^2} = -F\sin\theta\,, \tag{3.7}$$

where s is the curvilinear position along the beam and θ is the angle that the beam makes with the horizontal at that point. Integrating this equation gives a, f and F in terms of α. Solutions usually focus on the non-dimensional parameters a/L, f/L and F/F_c, where $F_c = \pi^2 EI/L^2$ is the Euler buckling force, to remove scaling and unit issues. For example for f/L (Douthe, 2007):

$$\frac{f}{L} = \frac{\sin\frac{\alpha}{2}}{K\left(\sin\frac{\alpha}{2}\right)}, \text{where } K(x) = \int_0^{\frac{\pi}{2}} \frac{1}{\sqrt{1-\sin^2 x \sin^2 \varphi}} d\varphi\,. \qquad (3.8)$$

Extracting a/L from physical observations or numerical simulations, and inverting the relationship on a/L, it is then possible to get the angle α corresponding to that beam, and the rest of the parameters. This means that for one simulation, there are three possible comparisons to the analytical values (a/L, f/L and F/F_c) from one measurement (on α).

3.3.2 Numerical simulations

The simulation model adopted for the beam is the same in each of the two solvers. The connectivity model stems from two straight lines of length $L/2$ forming an isosceles triangle with the horizontal axis as the base of the triangle. The lines are connected at the apex of the triangle and form an angle $\alpha = 1°$ with the horizontal. The lines are discretized in $n/2$ segments each, such that we always have an even total number of segments and a vertex in the middle of the discretized beam.

Internal forces are represented by elastic forces, as available in Kangaroo 1 and 2, see Figure 3.3, bottom. Position constraints are defined by an elastic linear spring of rest length 0 and given stiffness, attached on one side to a virtual fixed point and on the other to a defined vertex of the model. Distance constraints are represented by a linear spring of given rest length and stiffness, attached to two given vertices of the model. Angle constraints, representing bending forces, are represented by the discrete three points model from Adriaenssens and Barnes (2001), describing the shear force acting on the vertices when the interpolating arc going through them is bent.

In Figure 3.3, this action of the bending moment is represented by a pseudo-rotational spring, that will be integrated by the model into a relationship between bending angle and shear forces applied on the three nodes around it. k^* is the strength of this relationship as defined in Kangaroo, with units of rotational stiffness times length (force * length^2). This value has the advantage of being independent of the discretization length. For angles close to 180°, this is equivalent to a rotational spring of stiffness k^*n/L. Although this model does not directly track torsional effects, results remain valid for initially straight and untwisted uniform isotropic sections where torsion occurs as a result of out-of-plane loads (Adriaenssens & Barnes, 2001).

The left endpoint is connected to an elastic anchor of stiffness 10^{14} N/m. A distance constraint is added between the endpoints, with stiffness 10^8 N/m and rest length corresponding to the target a/L. Each segment is characterized by a distance constraint of stiffness (called "strength" in Kangaroo) EAn/L and rest length L/n. Each angle between two consecutive segments is characterized by an angle constraint with rest angle 0 and stiffness (or strength in Kangaroo) EI. The last two properties represent a physically correct discretized beam with the given E, A and I properties.

We used fixed values for L, E, A and I. We chose the tie stiffness so that it would be around one order of magnitude higher than EA. This way, the target length of the tie would be closely matched without introducing unnecessarily disparate stiffnesses in the model, which tend to make it less likely to converge. The stiffness of the anchor is high but is only used as a safeguard against the rigid-body movement of the model.

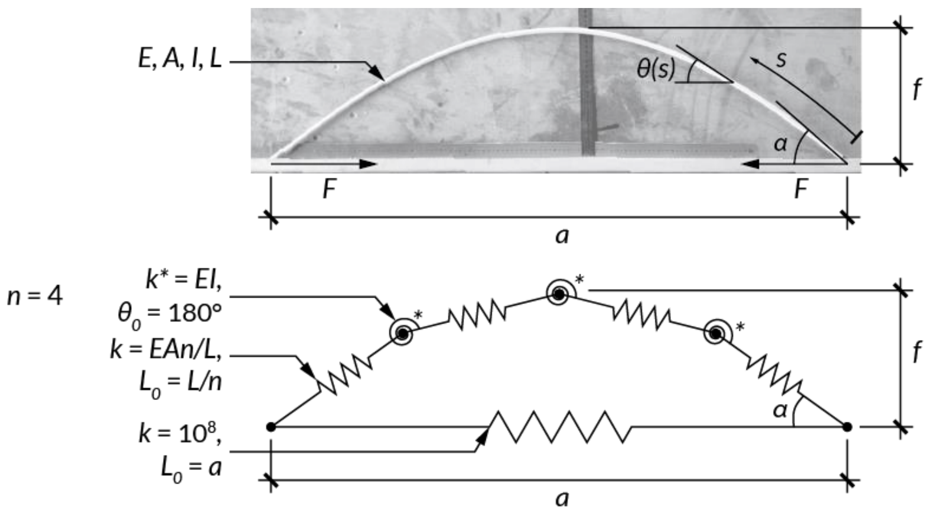


Figure 3.3: Definition of the planar elastica problem, continuous (top) and after discretization (bottom). The beam is pinned at both ends. The solution is a relationship between a, the distance between the supports, F, the reaction force at the supports, f, the maximum height of the beam over the support line, and α, the angle of the beam at the supports. See text for a description of the pseudo-rotational spring and definition of k^*.

3.3.3 Variables and observations definitions

From the nodal positions at the end of the simulation, we extract several observations:

- α , the angle of the first segment with the horizontal – this slightly underestimates the real initial angle of the beam interpolating through the vertices but is coherent with the discretized beam model;
- f, the distance from the middle vertex to the horizontal tie;
- a, the distance between the two endpoints;
- F, the support reaction, is computed by multiplying the tie stiffness by the difference between a and the tie's rest length.

These are the observations we compare against analytical results. Note that because the tie has a finite stiffness, its final length a is not exactly its rest length. Then we must make our analytical predictions based on the observed a, not on the tie's rest length.

In the analysis, we always present a comparison of observed simulations versus analytical results as an error on non-dimensional parameters. For example, for an observation f_{obs} associated with an analytical result f_{ana}, the "error on f/L" is:

$$\text{error}\left(\frac{f}{L}\right) = \frac{|f_{obs} - f_{ana}|}{|f_{ana}|}. \tag{3.9}$$

This helps in comparing errors across different boundary conditions, solvers, and observation types.

The stopping criterion terminates the simulation when the particles' total kinetic energy T falls below a fixed threshold, the final position is the equilibrium configuration. We checked that this happened before the solver reached its maximum number of iterations, in our experiments on the elastica. Kangaroo 2 does not use an explicit time step as Kangaroo 1 does, this changes how velocities are computed so the kinetic energy cannot be compared between the two. The number of iterations corresponds to the number of times the points were moved on the process of finding the equilibrium. The damping parameter used by Kangaroo 1's length constraints is chosen to be as close as possible to critical damping such that all results converge on a test case with $a/L = 0.74$. The time step used in Kangaroo 1 is the length of the time discretization interval, or time represented by an iteration.

The wall time is the time that the elastica problem took to run, as reported by the program launching the simulations. While wall time can be influenced by other routines running on the computer, efforts were made to minimize these effects during the simulations so that the results can be reasonably compared.

Table 3.1: Parameters used in the studies and presented in Section 3.5. [*start*:*step*:*end*] is used to represent the set of numbers from *start* (inclusive) to *end* (exclusive), stepping by increments of *step*. [*start*:*end*] = [*start*:1:*end*].

Solver	**Discretization**	**Compression ratio**	**Stopping criterion**	**Timestep**	**Damping parameter**
Kangaroo 1	$n \in [2:2:36]$	$\frac{a}{L} \in [0:0.02:1]$	$\log_{10} T \in [-12:-10]$	0.05	10
Kangaroo 2			$\log_{10} T \in [-14:-11]$	N/A	N/A
Custom dynamic relaxation			$\log_{10} T \in [-12:-10]$	0.05	10
L-BFGS			Energy: 10^{-7} Constraints: 10^{-4}	N/A	N/A
SLSQP					
Constants	$E = 10$ GPa, $R = 5$ cm, $L = 20$ m, $A = \pi R^2$, $I = \frac{\pi R^4}{4}$, $k_{Tie} = 10^8$, $k_{Anchor} = 10^{14}$				

Finally, the combination of parameters that were tested is represented in Table 3.1, totaling close to 40000 experiments. Scripts were created to automatically run the simulation in each case, time it, and collect the results.

3.3.4 Speed, accuracy, and reliability

Throughout this work, we use three concepts to evaluate the quality of the tools: speed, accuracy, and reliability. Speed is simply evaluated by the wall time (described above) taken by the algorithm. Speed is nothing if the result is not correct; thus we also look at accuracy, evaluated with the error measure defined in the previous subsection. This gives an upper bound achievable by the tool. A reasonable goal that is close to typical construction tolerances could be set at a 1% error. Finally, reliability is evaluated by the likeliness of the algorithm to output an incorrect solution. All the elastica results presented converged to the correct solution, but for example the introductory example of Section 3.1 shows that this is not always the case for more complicated structures.

3.4 Implementation details

In this chapter, we compare the pre-existing solutions used in the architectural form-finding community, to others borrowed from general non-linear optimization and computer graphics. Formally, form-finding methods are differentiated on two points: the bending energy discretization they use, and the optimization algorithm they follow. A possible implementation of these methods is described in this section.

3.4.1 Bending energy discretizations

3.4.1.1 Discretization based on a circular interpolant spline

Two discretization methods are generally used in similar problems. The first (Barnes et al., 2013) comes from the form-finding community and focuses on producing a simple expression for the forces acting on the vertices. It obtains these forces by considering an arc going through three consecutive points. For a series of three vertices $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$ forming an angle α, the resulting forces are (Adriaenssens & Barnes, 2001):

$$(\boldsymbol{S}_1, -(\boldsymbol{S}_1 + \boldsymbol{S}_3),\, \boldsymbol{S}_3), \qquad \text{where} \begin{cases} \boldsymbol{S}_1 = \dfrac{2EI \sin\alpha}{\|\boldsymbol{x}_2 - \boldsymbol{x}_1\|\|\boldsymbol{x}_3 - \boldsymbol{x}_1\|}\boldsymbol{n}_{12} \\ \boldsymbol{S}_3 = \dfrac{2EI \sin\alpha}{\|\boldsymbol{x}_3 - \boldsymbol{x}_2\|\|\boldsymbol{x}_3 - \boldsymbol{x}_1\|}\boldsymbol{n}_{23} \end{cases}, \tag{3.10}$$

and $\boldsymbol{n}_{ij}$ is the normal to the edge ij, in the plane defined by $(\boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3)$, pointing away from the center of curvature.

3.4.1.2 Discretization based on a discrete definition of curvature

Another method often referenced is the Discrete Elastic Rods (Bergou et al., 2008), see Figure 6 for notations. In this setup, the bending energy comes from a discretization of the curvature on a discretized rod:

$$E_{bending} = \sum_{i=0}^{n} \frac{EI(\kappa\boldsymbol{b}_i)^2}{\bar{l}_i}, \qquad \kappa\boldsymbol{b}_i = \frac{2\boldsymbol{e}^{i-1} \times \boldsymbol{e}^i}{\|\bar{\boldsymbol{e}}_{i-1}\|\|\bar{\boldsymbol{e}}_i\| + \boldsymbol{e}_{i-1} \cdot \boldsymbol{e}_i}, \tag{3.11}$$

where $\kappa\boldsymbol{b_i}$ is the curvature binormal vector at node i, that rotates the edge before node i into the edge after it, and $\bar{l}_i$ the length of the edges coming to node i. The bar designates initial values, that are kept constant for edge lengths.

This leads to the following forces (opposite of the bending energy gradient) acting on node i:

$$\boldsymbol{F}_i = \boldsymbol{F}_i^P + \boldsymbol{F}_i^A + \boldsymbol{F}_i^N \tag{3.12}$$

$$\boldsymbol{F}_i^P = \frac{2\alpha}{\bar{l}_{i-1}}\left[\frac{2\boldsymbol{e}_{i-2} \times (\kappa\boldsymbol{b})_{i-1} + \|\kappa\boldsymbol{b}\|_{i-1}^2\boldsymbol{e}_{i-2}}{\|\bar{\boldsymbol{e}}_{i-2}\|\|\bar{\boldsymbol{e}}_{i-1}\| + \boldsymbol{e}_{i-2}\cdot\boldsymbol{e}_{i-1}}\right] \tag{3.13}$$

$$\boldsymbol{F}_i^A = -\frac{2\alpha}{\bar{l}_i}\left[\frac{2(\boldsymbol{e}_{i-1} + \boldsymbol{e}_{i+1}) \times (\kappa\boldsymbol{b})_i - \|\kappa\boldsymbol{b}\|_{i-1}^2(\boldsymbol{e}_{i-1} - \boldsymbol{e}_{i+1})}{\|\bar{\boldsymbol{e}}_{i-1}\|\|\bar{\boldsymbol{e}}_i\| + \boldsymbol{e}_{i-1}\cdot\boldsymbol{e}_i}\right] \tag{3.14}$$

$$\boldsymbol{F}_i^N = \frac{2\alpha}{\bar{l}_{i+1}}\left[\frac{2\boldsymbol{e}_{i+2} \times (\kappa\boldsymbol{b})_{i+1} - \|\kappa\boldsymbol{b}\|_{i+1}^2\boldsymbol{e}_{i+2}}{\|\bar{\boldsymbol{e}}_{i+1}\|\|\bar{\boldsymbol{e}}_{i+2}\| + \boldsymbol{e}_{i+1}\cdot\boldsymbol{e}_{i+2}}\right] \tag{3.15}$$

In this chapter, when relying on custom implementations, we use the second definition of forces, as it tends to give better results when coarser discretizations are used.

3.4.2 Software implementation

While the general framework built for comparing these methods is written in Python, a language that generally does not have the speed of compiled languages such as C++, we took great care to use efficient routines for all time-critical code. All data structures are represented using Numpy arrays, that use compiled linear algebra routines for all mathematical operations. Wherever possible, we used a sparse representation of the data structures, especially for the edges connectivity matrix and the constraints enforcement schemes that require sparse least-squares solves; we found a typical speedup of 3-5x using that. For the computation of constraints residuals, energies, and forces, we used code compiled using the Numba compiler for Python, and leveraged sparsity information when possible; this leads to 10-100x speedups in the implementation. Finally, all algorithms call compiled routines for their main work: the SLSQP, L-BFGS, and augmented Lagrangian implementations used are shipped with Fortran routines, and the dynamic relaxation code was compiled using Numba; this last operation gave 10x speedups.

3.5 Results of numerical experiments

In this section, we present the results of our solvers, on the set of numerical experiments described above, focusing on speed and accuracy. Then, we consider a larger example solved using Kangaroo 2 only. There is no analytical solution in that case to evaluate the accuracy of the solvers, but it displays reliability behavior that is of interest in selecting the stopping criterion.

3.5.1 Kangaroo 1: dynamic relaxation

The first set of results we present in Figure 3.4 are from Kangaroo 1. They show the error on three different measures, in log scale, as we vary the number of segments n and the compression ratio a/L. The errors represented are on α, F/F_c and f/L. The stopping criterion used was 10^{-10}, this value displays similar behavior to smaller thresholds as shown in Figure 3.5.

The three graphs show similar patterns: for small n, the error is close to 100%, meaning that the model failed to predict reality; then the error decreases asymptotically towards 0.1% as n is increased. This is only true in a limited range of a/L: for a/L smaller than 0.2 the model rarely represents reality, even as n reaches 36. This shows that Kangaroo 1 can only represent limited amounts of bending (when a/L is high, the beam is close to a flat line), and should only be used when the elasticas are compressed by less than 50%.

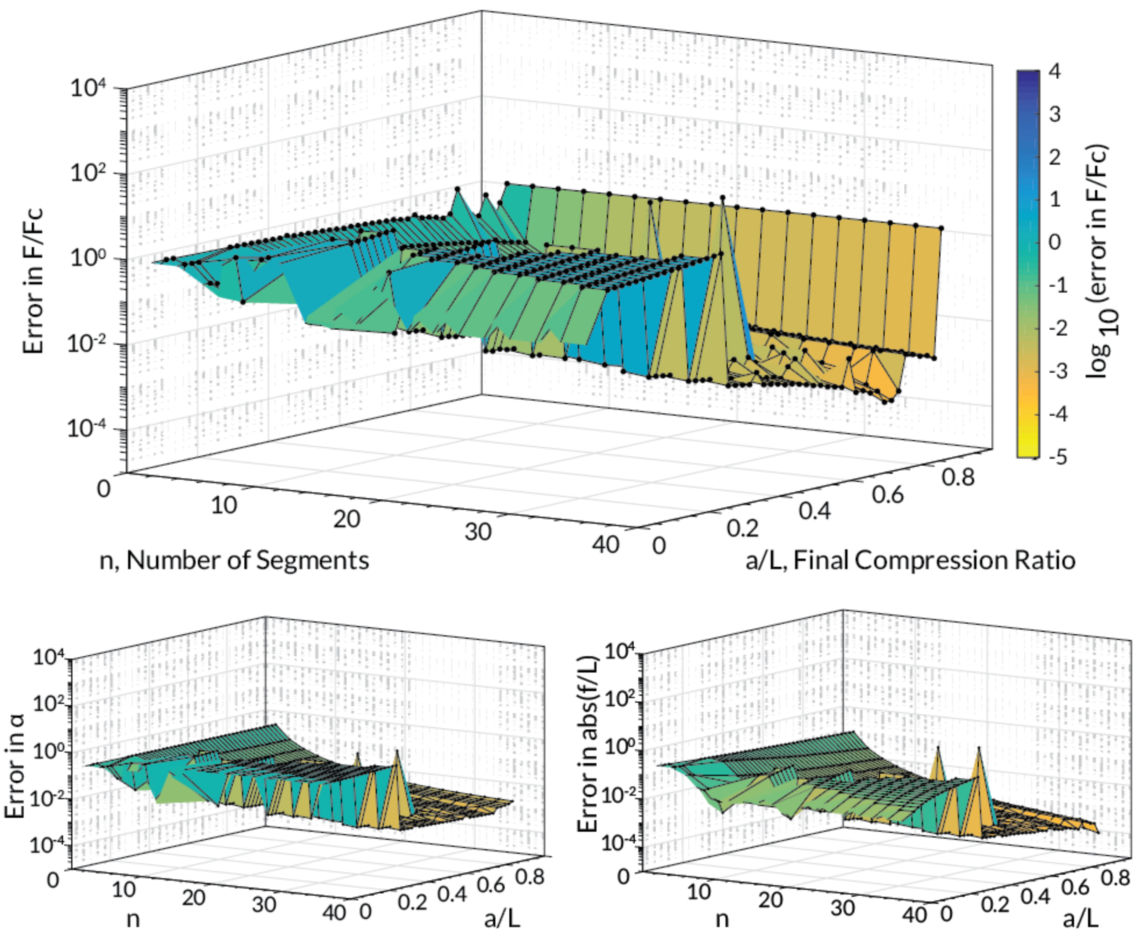
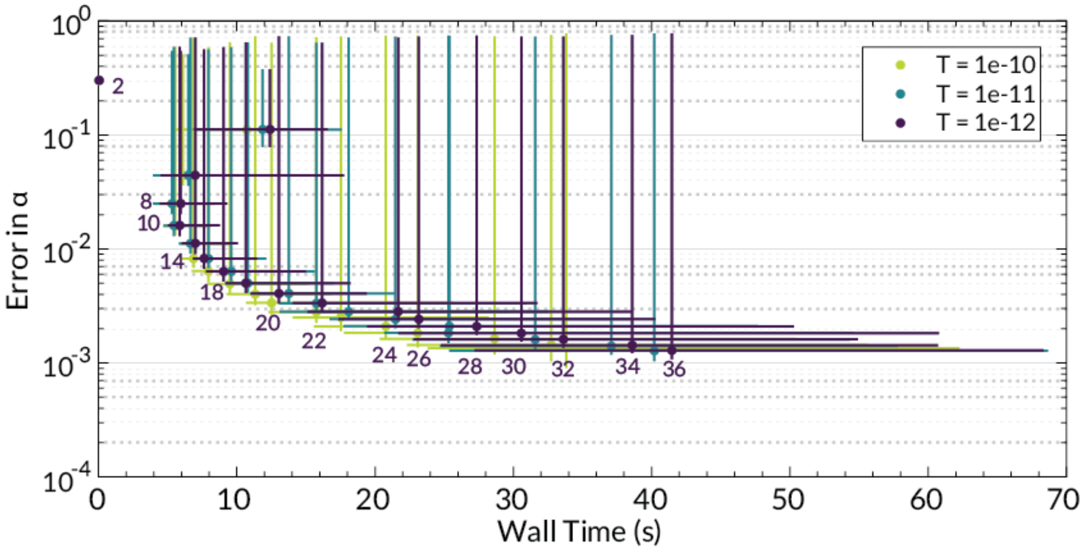


Figure 3.4: Surface plots of errors in Kangaroo 1 simulations of elastica for different numbers of discretized segments and compression ratio. Kinetic energy threshold used: 10^{-10}.



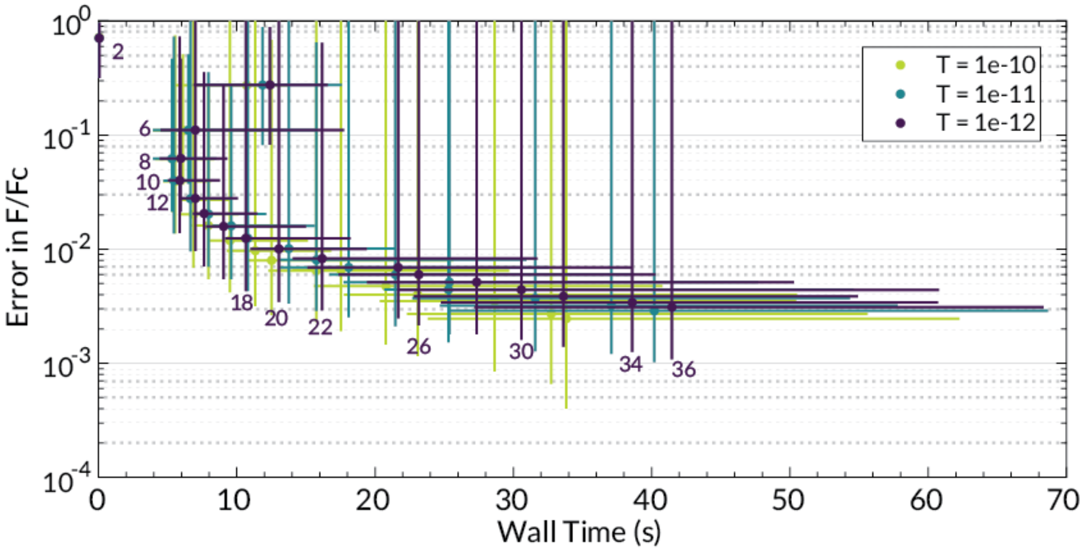


Figure 3.5: Time vs. error in Kangaroo 1 for elastica simulations. Each color represents one threshold for the stopping criterion, each point represents one number of discretized segments (labeled). The point is at the median time and median error for all simulations that have the same number of segments and the same threshold. The extent of the bars represents the spread from 1st to 9th decile in time and error for these same simulations.

In the error on F/F_c, the points that have $a/L = 1$ have a high error. This is because, at these points, the solution beam is a nearly flat yet buckled beam, while in reality it should be exactly flat at the onset of buckling. Then the force derived from Kangaroo 1 is largely underestimated.

In general, Kangaroo 1 is complex to run reliably, often reaching divergent conditions. Figure 3.5 shows that the only way to achieve a precision of 1% or better in F/F_c is with $n \approx 20$, which takes 10 to 20 seconds to run for one single elastica. Even with small timesteps, the simulation is not reliable as a large proportion of the simulations have errors close to 1. This is true for several values of the stopping criterion, it even seems that increasing it shows, in general, a degradation of accuracy.

3.5.2 Kangaroo 2: projective dynamics

Next, we present similar studies for Kangaroo 2. The stopping criterion used was 10^{-12}, Figure 3.6. Figure *3.7* shows how it compares to other possible choices. The plots show the same global trend on n, with the error going from 1 to 0.1% on average when n is increased, in the errors on α and f/L. The error on F/F_c initially starts at much higher levels, but then quickly returns to usual when $n > 6$. Here, we find no evidence of particularly unstable regions in a/L, except on F/F_c when $a/L = 1$, as seen before. This shows that the solver is more reliable, as the error is almost constant across a wide range of boundary conditions. The error tends to be less constant when $n > 24$, especially in α and f/L, but in general remains bounded under the general trend going towards 0.1% error.

In general, it seems that around 20 elements are needed to reliably get an error below 1% in F/F_c. In the other two errors this is achieved even for $n = 10$. Figure *3.7* confirms this behavior, showing that it is possible to achieve a 1% accuracy in 0.5 seconds. This is using 14 to 18 elements per elastica, with a threshold of 10^{-12} to 10^{-14}. However, it seems for thresholds of 10^{-12} or higher, increasing the number of elements risks reducing the accuracy. Additionally, accuracies better than 0.1% are almost impossible to obtain reliably for a range of boundary conditions.

In general, these studies show that Kangaroo 1 is not appropriate as a reliable design tool for bending-active structures, as no combination of parameters allows it to reach a predictable level of accuracy or speed. Kangaroo 2, on the contrary, is a good candidate for rapid design iterations in bending-active structures. For a typical rod it converges in less than a second to accuracies of 1% or better in position and forces. We recommend using a threshold of 10^{-12} or smaller, and 15 to 20 segments per discretized beam. Table 3.2 summarizes these results.

Table 3.2: Summary of results for the elastica experiments.

Software	**Threshold for a 1% accuracy**	**Typical run time for a 16 nodes problem**
Kangaroo 1	10^{-10}	10 seconds
Kangaroo 2	10^{-12}	0.5 seconds

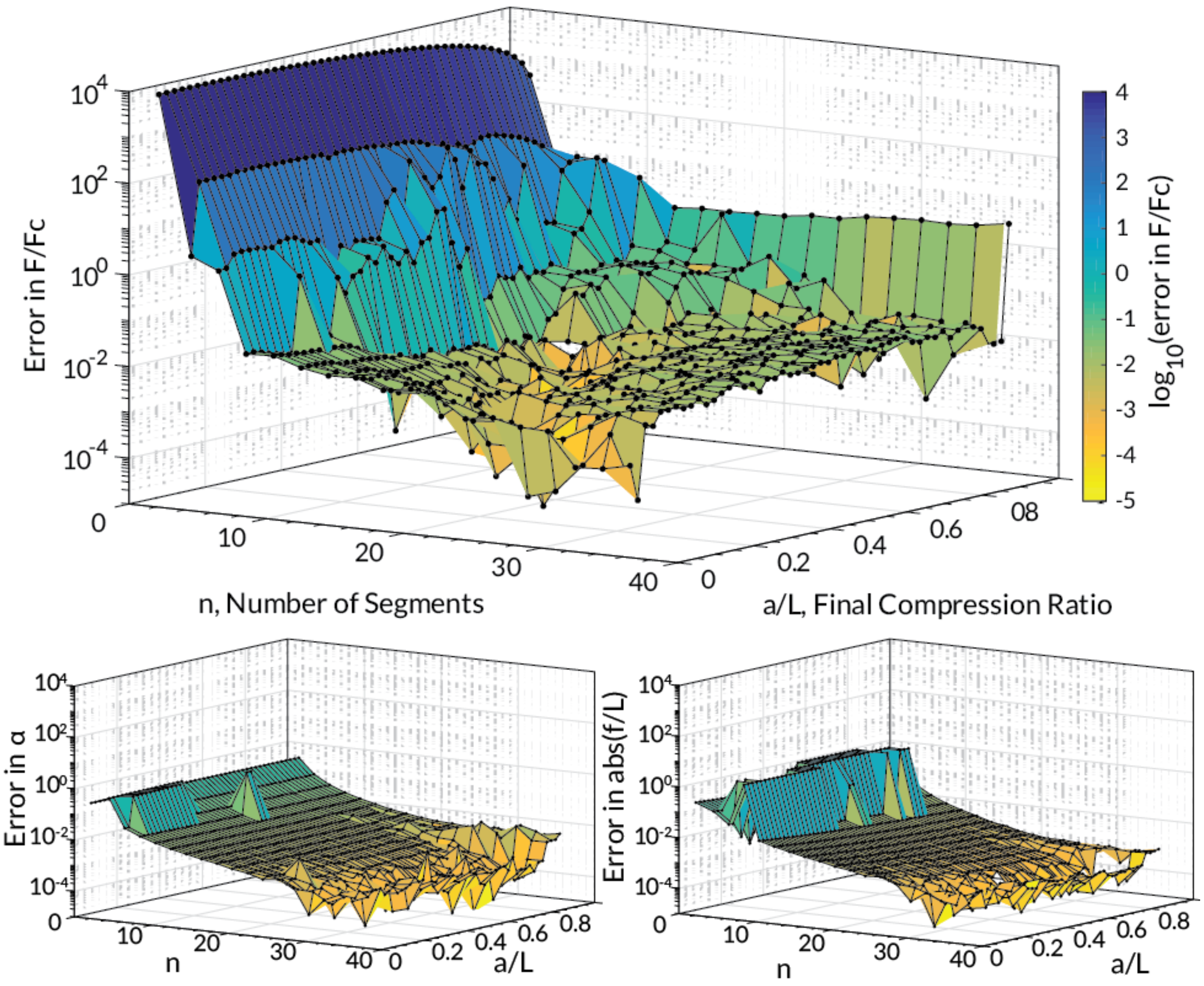


Figure 3.6: Surface plots of errors in Kangaroo 2 simulations of elastica for different numbers of discretized segments and compression ratio. Kinetic energy threshold used: 10^{-12}.

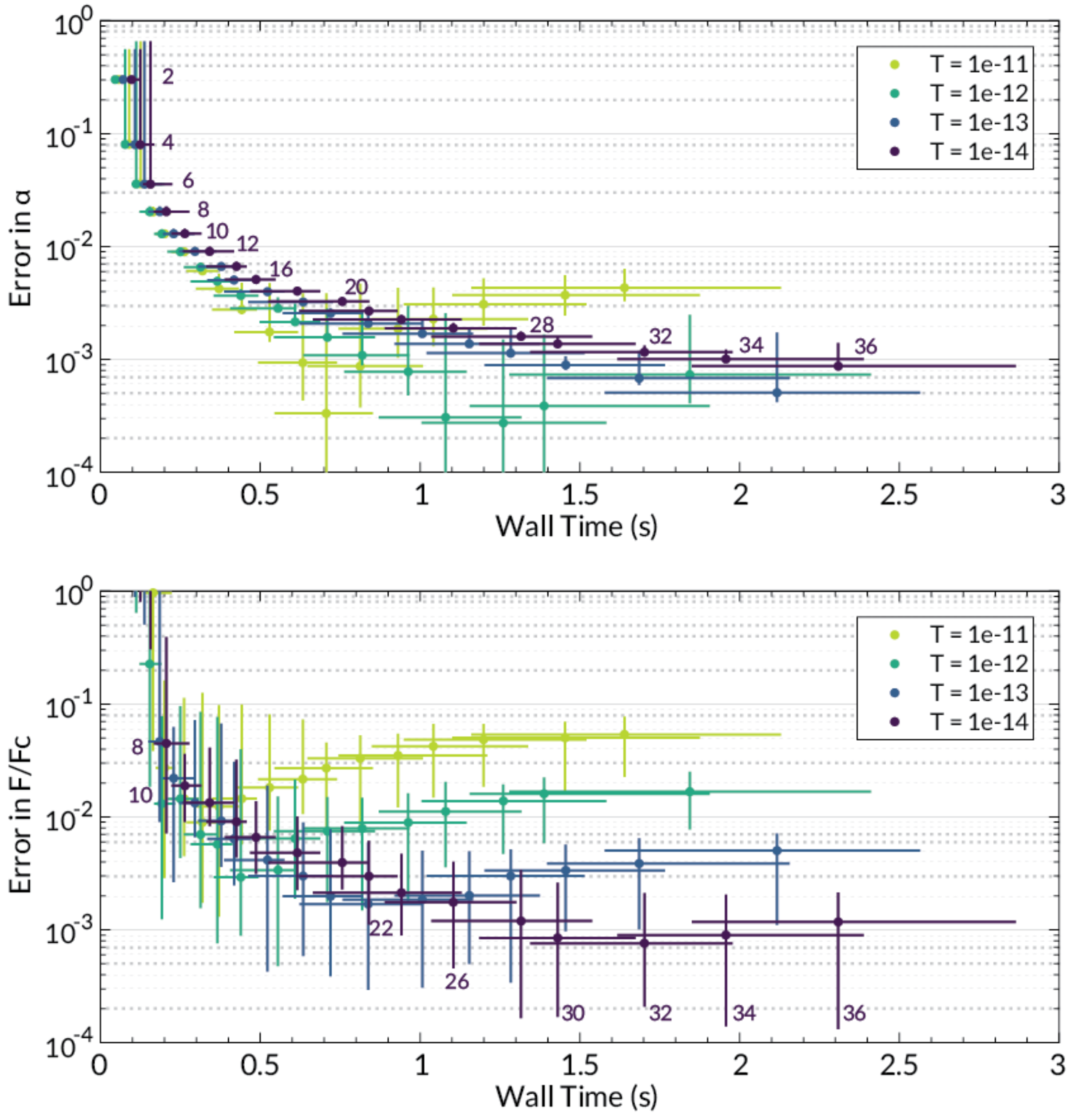


Figure 3.7: Time vs. error in Kangaroo 2 for elastica simulations. See Figure 5 for labels.

3.5.3 Comparison with general-purpose optimizers

In this section, we present the comparison of observed simulations results versus analytical results as an error on non-dimensional parameters. For example, for an observation of the height at the midpoint of the elastica f_o associated with an analytical result f_a, the relative error on f/L is: $e(f/L) = |f_o - f_a|/|f_a|$. This helps in comparing errors across different boundary conditions, solvers, and observation types. For all three solvers (SLSQP, L-BFGS in augmented Lagrangian, dynamic relaxation), we varied the solver parameters, aiming to find a combination that would reliably give a 1% accuracy on the relative error on f/L in the shortest runtime, for a collection of test cases presented in Table 3.1.

The stopping criterion terminates the simulation when the relative change in the elastica's energy is less than an "energy tolerance" between two iterations of the optimization procedure, and when the maximum of the constraints' residuals is less than a "constraints tolerance". We checked that this happened before the solver reached its maximum number of iterations, in our experiments on the elastica. The wall time is the time that the elastica problem took to run, as reported by the program launching the simulations. While wall time can be influenced by other routines running on the computer, we averaged each runtime over three runs, so that results are more reliable. For dynamic relaxation, we used a time step that was close to one-tenth of the fundamental vibration period between two successive edges.

The results are presented in Figure 3.8 and Figure 3.9. For SLSQP and L-BFGS, we only present the combinations that gave the best results. For dynamic relaxation, the results were less clear so we present them for three values of the energy tolerance. SLSQP and L-BFGS fare very well on these elastica experiments, with the 1% accuracy threshold reliably obtained with as little as 9 points on the elastica. This is a lot less than with Kangaroo 1 or 2, where around 20 points were needed; this confirms the better convergence properties of the energy model in 3.4.1.2 versus 3.4.1.1. Increasing the number of vertices to larger values tends to give less reliable results, usually because the solver found a local minimum of energy and stopped too early. Lowering the energy threshold tended to improve on this point. L-BFGS is slower than SLSQP for small numbers of vertices, this is mostly due to a longer startup time, and L-BFGS catches up quickly as this number is increased.

Dynamic relaxation is harder to get to converge reliably, and we found the best results for an energy threshold of 10^{-10} to 10^{-11}, with 9 to 11 vertices. This is coherent with the analysis using Kangaroo 1 and 2. Our implementation of dynamic relaxation is around 10 times slower than SLSQP and L-BFGS, although it is still 100 times faster than the implementation in Kangaroo 1. This shows the dramatic influence that the enforcement scheme on the axial constraints can have.

Both dynamic relaxation and L-BFGS exhibit unstable behaviors when the number of elements grows too high. This is because this generates a stiffer system that needs stricter parameters in the solver to be solved reliably (for example for dynamic relaxation, a smaller time step). Because we aim to find just one set of solver parameters and the number of discretization elements that will find an accurate solution (less than 1% error) in the shortest amount of time, these results show that this combination of parameters should not be used in our case.

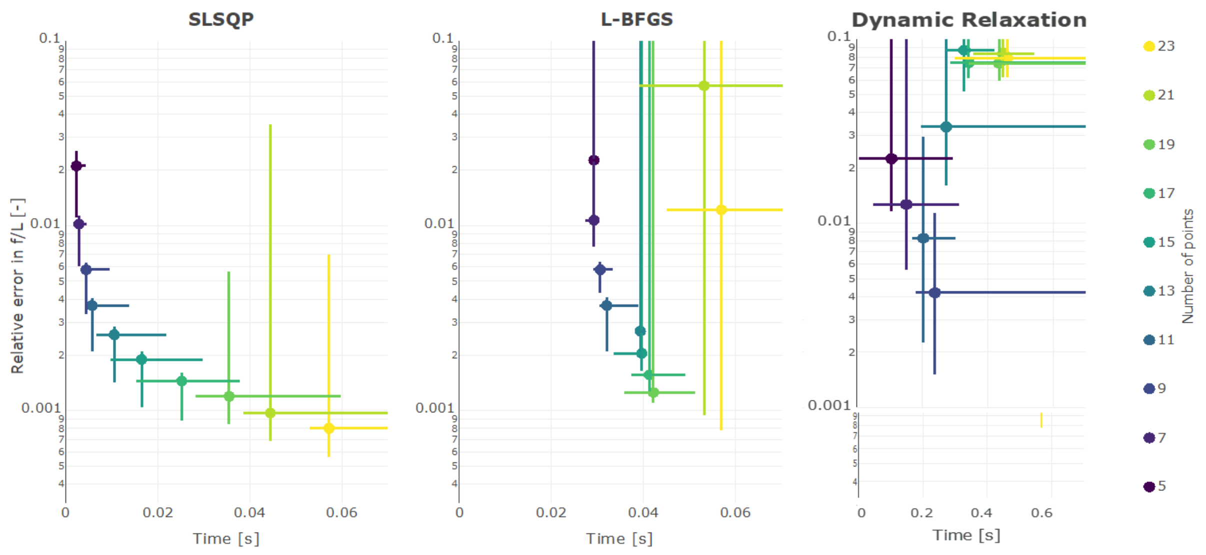


Figure 3.8: Time vs. error in height with SLSQP and augmented Lagrangian L-BFGS elastica simulations, for an energy tolerance of 10^{-7} and a constraints tolerance of 10^{-4}. Each point represents one number of discretized segments. The point is at the median time and median error for all simulations that have the same number of segments and the same tolerances. The extent of the bars represents the spread from 2nd to 8th decile in time and error for these same simulations.

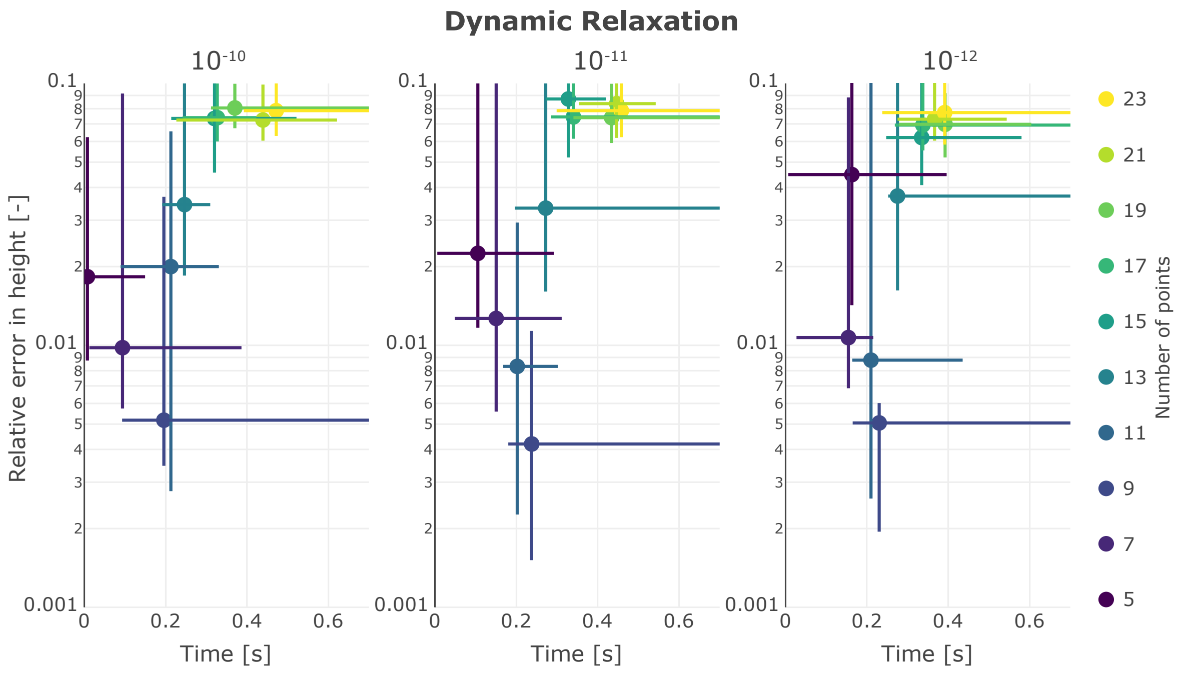


Figure 3.9: Time vs. error in dynamic relaxation with three values of energy tolerance (labeled), constraints tolerance of 10^{-4}, for elastica simulations. See Figure 3.8 for labels.

3.5.4 Larger examples

3.5.4.1 Cocoon

We applied our recommendations to the *a posteriori* simulation of a bending-active structure. The Cocoon project, built in Winnipeg, Manitoba, is a 12-meter-long pavilion made for the “Warming Huts” competition of 2012 (Coar, 2012). It is made of fiberglass rebars tied into “double-A-frame” modules and anchored into a frozen river. Dimensions and material properties are given in Figure 3.10.

A comparison of the numerical model to actual photos in Figure 3.11 shows good agreement in shape. We obtained this result by discretizing the beams in 15 elements each, using ties of stiffness 10^7 N/m and rest length 0 m, and anchoring with stiffness 10^9 N/m. Figure 3.13 (left) shows this assembly for one module of the structure. We ran the simulation in Kangaroo 2 for 10^4, 10^5 and10^6 iterations, with 10^5 iterations the closest to our recommended threshold of 10^{-12}, as demonstrated in Table 3.3. We found that 10^5 gave the best ratio of accuracy to time of computation. Comparing nodal positions to the reference run of 10^6 iterations, 10^5 iterations is a significant improvement over a threshold of 10^{-11} (10^4 iterations), without being far from the result found with 10^6 iterations, as shown in Figure 3.13. We chose to use the number of iterations as a stopping criterion instead because sometimes a threshold is never reached, see Table 3.3.

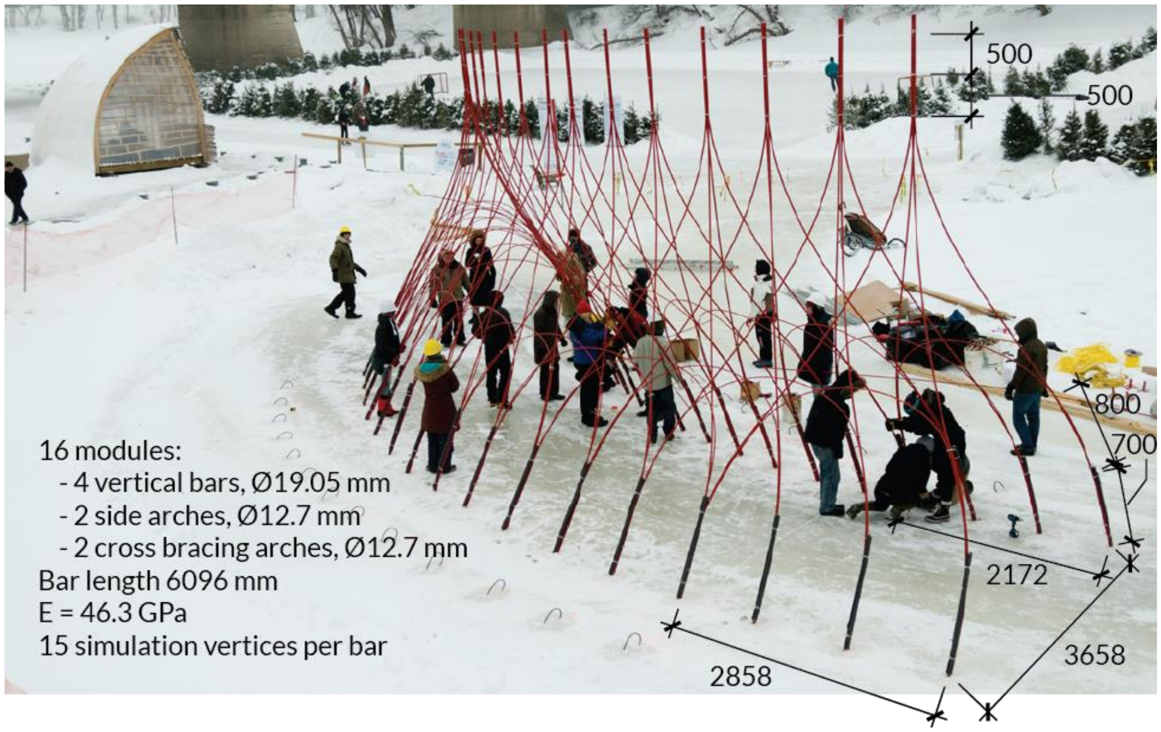

500
500
800
700
2172
3658
2858
16 modules:
- 4 vertical bars, Ø19.05 mm
- 2 side arches, Ø12.7 mm
- 2 cross bracing arches, Ø12.7 mm
Bar length 6096 mm
E = 46.3 GPa
15 simulation vertices per bar

Figure 3.10: Cocoon project (Coar, 2012): dimensions and material properties. Labeled length dimensions are in mm. Photo © Matthieu Léger.

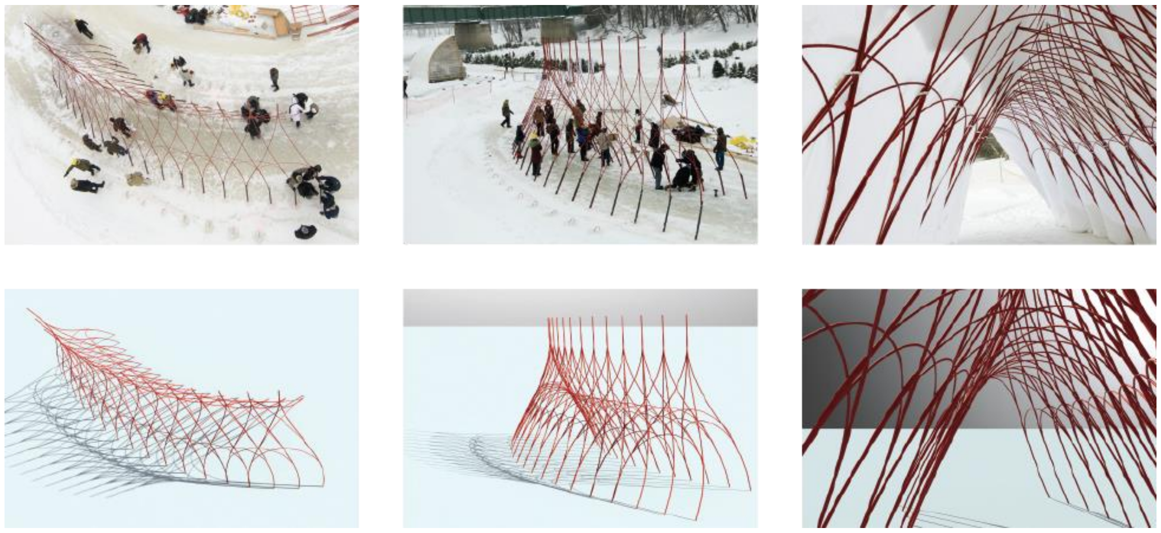


Figure 3.11: Comparison of actual footage of the construction to numerical simulations. Photos in top row © Matthieu Léger.

To check the quality of our model, we compared the simulated distance from the top points of the modules to their four anchors, to the same data extracted from photographs with different viewpoints. The results are presented in Figure 3.12, grouped by anchor position in the module and the number of iterations used. Across all measurements but one, the absolute value of the relative error is below 2.5 %. There is a clear improvement in errors from 10^4 iterations to 10^5 iterations, with the error getting below 1 % on almost all measurements, then very little change as the number of iterations is changed to 10^6.

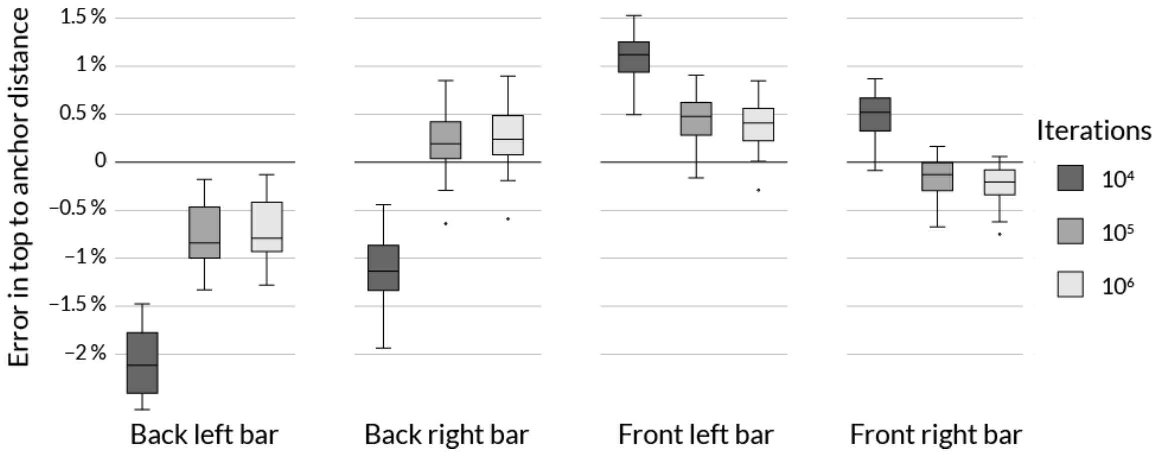


Figure 3.12: Boxplot of the relative errors in the simulated distance from the top points of vertical bars to their anchors, compared to the physical structure, grouped by position of the anchor and number of iterations. Anchors positions are the four corners of each module, as seen in Figure 8. The central bar in the box shows the median of the group of measures, the extent of the box shows the first and third quartile, the whiskers show the minimum and maximum data values, and the points are outliers.

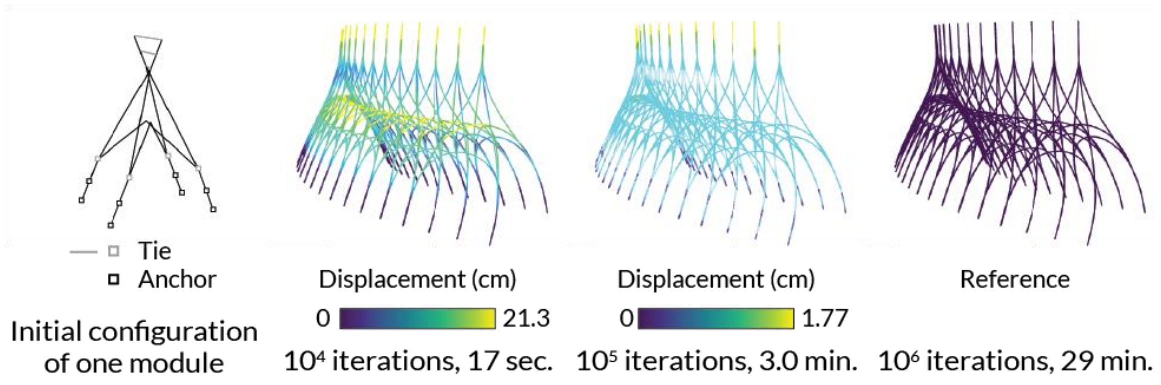

Tie
Anchor
Initial configuration of one module
Displacement (cm)
0
21.3
10^4 iterations, 17 sec.
Displacement (cm)
0
1.77
10^5 iterations, 3.0 min.
Reference
10^6 iterations, 29 min.

Figure 3.13: Comparison of results in positions for different numbers of iterations. Displacement shows the positional difference with the reference obtained after 10^6 iterations. In the initial configuration, point-like ties are connected to neighboring modules.

This is an encouraging result showing that our recommendations from the previous subsection work for larger scales of projects. We also find that this is true for reaction forces prediction; see Figure 3.14. However, Figure 3.15 shows that caution is still needed when external forces are applied. In this case, we applied a uniform moderate wind pressure of 0.2 kN/m^2 on the structure, in the pushing direction on the longer side, and pulling on the shorter side. As the simulation with 10^6 iterations shows, the structure is failing by buckling on the three external modules on each side (the inner modules are stiffer because they are connected to more neighbors). This does not happen in the shorter simulation runs, with the 10^5 iterations run only hinting at the phenomenon. This buckling behavior was also observed in other experimental prototypes of the structure.

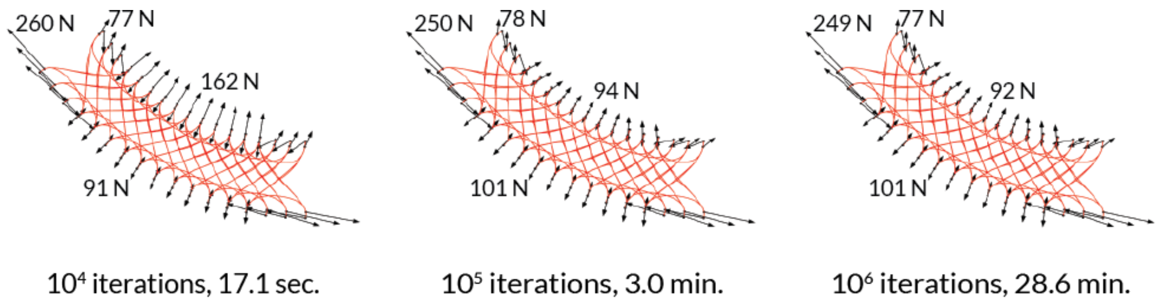

260 N
77 N
162 N
91 N
10^4 iterations, 17.1 sec.
250 N
78 N
94 N
101 N
10^5 iterations, 3.0 min.
249 N
77 N
92 N
101 N
10^6 iterations, 28.6 min.

Figure 3.14: Comparison of results in anchor forces for different numbers of iterations (top view). Maximum force vector error from 10^4 to 10^6: 76%, from 10^5 to 10^6: 1.5%.

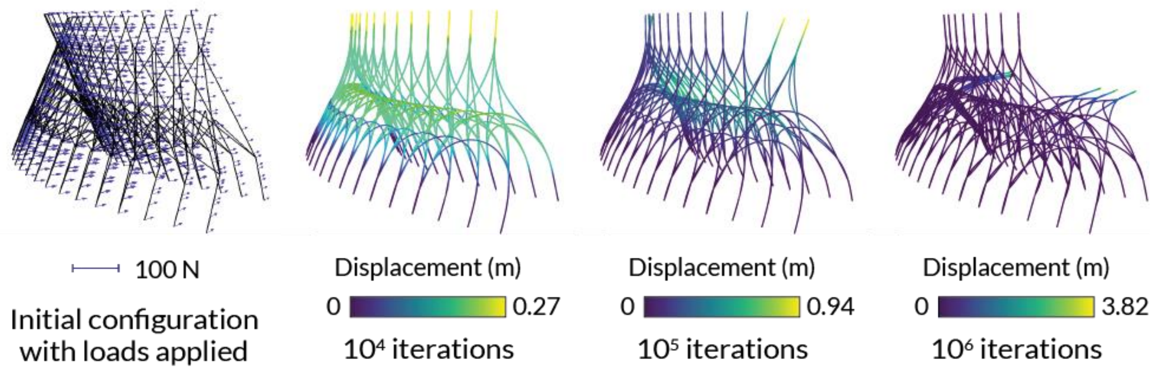


Figure 3.15: Comparison of results in positions for different numbers of iterations, when a uniform wind pressure is applied to the structure. The displacements are shown from the unloaded configuration with 10^6 iterations.

Table 3.3: Iterations needed for different energy thresholds in the Cocoon model. The 10^{-13} simulation did not converge after 12 hours of runtime.

Threshold	10^{-10}	10^{-11}	10^{-12}	10^{-13}
Number of iterations needed	6580	10320	25810	>30M

3.5.4.2 Elastic gridshell

Next, using an energy tolerance of 10^{-7}, and a constraints tolerance of 10^{-4}, we used the same algorithms to find the equilibrium position of an elastic gridshell, as represented in Figure 3.16, and varied the number of points between two connections of the gridshell from 0 to 4. This varied the number of points in the study from 94 to 654. Although all algorithms found a reasonable solution for the model with 94 points, the initial position in the model with more points proved too stable for dynamic relaxation and SLSQP, and only L-BFGS found good equilibrium positions in this case. Table 3.4 summarizes these results. It shows good convergence of the positional accuracy, and a roughly linear increase in runtime with the number of points, making the algorithm usable for larger models.

Since the equilibrium positions found by SLSQP and dynamic relaxation with 94 points have a positional accuracy of better than 1%, it is arguable that this model already has a sufficient number of points and both methods could then be used for a rough form-finding. In this case, SLSQP converged in 0.69 s and dynamic relaxation in 2.2 s, significantly slower than L-BFGS. Additionally for SLSQP and dynamic relaxation, the runtime per iteration is roughly quadratic in the number of points, indicating that these algorithms will become unusable for models with more than a few hundred points. This suggests that L-BFGS is the better choice for larger models.

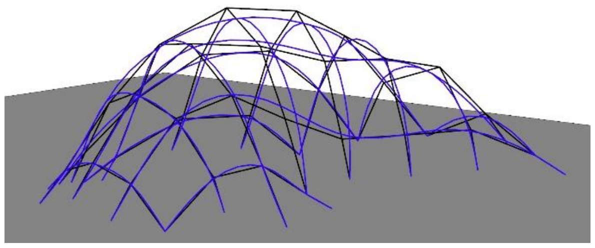


Figure 3.16: Gridshell example used. Initial positions in black, best equilibrium found in blue.

Table 3.4: Runtimes for L-BFGS with varying number of points in the beams. Accuracy refers to the mean distance between nodes and the closest point in the equilibrium position found with the highest number of nodes, divided by the size of the model.

Number of points	94	174	334	654
Runtime [s]	0.165	4.33	38.0	105
Accuracy	0.7%	0.08%	0.03%	-

3.6 Conclusion

This chapter has presented new results and guidance for modeling bending-active structures, focusing on key performance metrics: accuracy and speed. Our results show that designers cannot reliably use Kangaroo 1 for simulating such structures, but that Kangaroo 2 can produce good results when tuned properly. In general, there is a tradeoff between accuracy and speed, on a single elastica with Kangaroo 2 the error is at best 0.1% for a 1-second runtime. Some combinations of simulation parameters lead to long simulation times and high errors in geometry and forces, especially when thresholds are set too low. This can impede the creative design process both by interrupting a designer's flow and by misrepresenting physical reality. We recommend using in Kangaroo 2 a threshold of 10^{-12} or smaller, and 15-20 nodes per element in bending, to achieve a spatial accuracy of 1% or better. This is when using the physical values EA for the axial stiffness and EI for the bending stiffness, which ensures that positions and forces extracted from the model can be directly linked to the physical structure.

Using the guidelines proposed here, the Cocoon case study illustrates that high-quality simulation of bending-active structures is possible, but not guaranteed with contemporary, widely available tools. Because the simulation is so sensitive to modeling parameters, critical behavior such as buckling under wind loading can be missed. Furthermore, the physical structure will be impacted by construction tolerances and site variables. As a result, the usefulness of a precise digital model will depend greatly on an equally precise and constrained construction method that can reflect these accuracies.

The results give a better understanding of how computational design tools for bending-active simulations work and perform for physically realistic modeling. They show the relevance of generalist optimization methods such as SLSQP and L-BFGS for these simulations, even compared to methods well-established in the field like dynamic relaxation. Finally, they offer guidance on possible future developments for efficient design tools and the accurate use of existing tools.

In particular, using L-BFGS with a stopping criterion on the relative change in energy of 10^{-7}, a constraints tolerance of 10^{-4}, and at least 9 points per beam seems to be both stable and time-efficient for small and larger models. On small models (less than a hundred points), SLSQP with the same parameters is competitive in time, and sometimes more stable. In terms of discretization techniques, it seems that an approach such as Bergou et al. (2008) that represents both length constraints and bending energy in a unified framework is beneficial to the accuracy of the complete software.

In future work, it would be interesting to see how these results evolve when more complex beam models are considered, for example including torsion effects. It seems it would also be beneficial to use optimization algorithms derived from SLSQP such as (Kovalsky et al., 2016), which can use sparsity information in the problem, mitigating the quadratic increase in runtime with the number of nodes.

Future research is needed to understand different bending-active typologies, such as those that use hollow tubes instead of rods as elasticas. In this case, different ratios of bending to stretching stiffness are present, so this work's results may not be directly applicable. In general, simulating tube-based bending-active structures should be faster because of a smaller range of stiffnesses, but not necessarily more accurate.

In closing, the exciting possibilities of bending-active structures are expanded by the proliferation of designer-accessible simulation tools such as Kangaroo 1 and 2. However, as shown in this work, a better understanding of how these tools work and perform is needed for physically realistic modeling, and trust in seemingly precise digital results can be easily misplaced. The results presented in this chapter contribute clarity in this direction and offer steps for improved workflows for designing bending-active structures. This also paves the way for better inverse form-finding procedures, where the speed and robustness of the inner loop do not hinder the optimization process of the outer loop.

4 Inverse form-finding for bending-active structures

Finding a bending-active structure that is as close as possible to a target shape is an inverse form-finding problem, that can be solved using a nested optimization process. In the outer optimizer, we iterate on some initial condition of the bending-active structure, until the distance from the equilibrium shape given the current initial conditions is as small as possible. Finding the equilibrium position is the result of the inner optimizer. This inner optimizer needs to be stable, reliable and fast, given the wide variety of input conditions that the outer optimizer will attempt. This chapter uses the results from Chapter 3 to build such a solver, and apply it to two design problems: a simple arc-lamp first, and a larger elastic gridshell second.

Because this chapter reuses the solvers and methods studied in Chapter 3 in the inner loop of the inverse form-finding process, the results of that chapter on the quality of the results produced still apply here: generally as accurate as other direct form-finding methods, and selected to be as fast as possible in these conditions.

4.1 Background

4.1.1 Bending-active structures

Shell structures derive their structural efficiency from their geometry. While their form minimizes bending moments, they are complex to build. Bending-active is a construction method to create curved shell geometry by elastically bending initially straight rods. It allows the creation of shell-like shapes with a simpler construction process, leading to lightweight structures with possibly long spans and quickly erected. Examples of such constructions include the well-known elastic gridshell of the Mannheim Multihalle by Frei Otto (1974, (Happold & Lidell, 1975)), and many others since then with regular (Merrick, 2006) and irregular arrangements of the rods (Coar, 2010), different materials like GFRP for the rods (Baverel et al., 2012) and concrete for the cover (Cuvilliers et al., 2017), or plate-like base elements (Nabaei et al., 2013).

Automatically predicting the equilibrium shape of the bending-active structure at the end of this process is critical to enabling their design, as it is very difficult to intuit the shapes they can take. In addition, no good criterion exists for predicting if a surface can be approximated by a given arrangement of rods, or which arrangement would give the best result. As a result, all design workflows start by simulating the equilibrium shape of the structure considered, a process that can be time-consuming and error-prone, before analyzing the qualities of the resulting design.

By automating this process, carefully tuning it for speed and reliability (Chapter 3), and integrating it in an optimization loop, the designer gains access to a design workflow where objectives can be specified for the design. The rod arrangement, lengths, and properties needed to reach these objectives is the result of this workflow. Most importantly, this lets designers come as close as possible to a target shape that they define.

4.1.2 Target shape

Computational form-finding methods usually output forms that minimize some part of the stresses in a given structure, for example shell bending stresses in the design of compression-only shells. In bending-active structures, the normal form-finding process finds the equilibrium shape by minimizing the bending energy in the structure. It is also possible to further optimize the resulting structure to reduce stresses under live loads or buckling sensitivity (D'Amico et al., 2015).

While this helps reduce the material quantities and increase the performance of the shell structure, the designer has limited control over the resulting shape. By defining a desired target shell shape, structural optimization tools can be successfully applied to find forms that both minimize structural material volumes and resemble the target shape. Modifying the equilibrium shape is most easily achieved by varying the cross-section of the rods. While this does not apply to all materials, timber lends itself quite well to varying cross-sections (Mayencourt et al., 2017).

Optimizing for a target shape is one of the most evident inverse form-finding problems, especially for bending-active structures where the equilibrium shape can exhibit dramatic variations when the initial conditions are changed. The target shape is a simple expression of the designer's intent and makes for a simple objective formulation. Garg et al. (2014) use a similar approach to build a design tool for sculptures made of wire meshing, a closely related problem but where the elastic bending forces can be neglected as the material is plastically deformed to conform to the target surface. This simplifies finding the equilibrium shape of the structure, and most of the attention is given to the outer optimizer. The authors use a two-part objective function, summing a “fairness” and a “closeness” term. The closeness is simply a Euclidian distance from the nodes of the mesh to their closest projection on the target; the fairness represents the amount of curvature in the form-found surface, minimizing it helps to avoid sharp features that are difficult to build. The variables in this setup represent the shape of the boundary of the initial mesh piece, the optimizer in effect is adding and removing material to use in the construction.

Similar design problems include the design of 3D-printed network of elastic rods that approximate a surface (Pérez et al., 2015; Zehnder et al., 2016). By deferring finding the elastic equilibrium until later in the design process, the inverse form-finding problem of finding the realizable structure closest to the target surface is more tractable. The objective again includes a Euclidian distance from the nodes of the simulation to the target surface, and additional terms representing relevant design objectives. Rao et al. (2019) for example add a term to represent a consistent pressure applied by a cast on a broken and swollen limb.

More recently, a complete design system was proposed, solving an inverse form-finding problem towards a target shape while simulating elastic bending equilibriums (Panetta et al., 2019). The structures built there are close to elastic gridshells, except that they do not start from a regular grid, and are actuated into their final shape by force actions on a small set of nodes, rather than by fixing the boundary. Their methodology validates many of the choices proposed in Chapter 3: the beam model is taken from Bergou et al. (2008), the inner optimizer is a Newton-based with Hessian information, similar to BFGS. The outer optimizer is the Newton-CG trust region method.

4.1.3 Design variables

In this chapter, we present two ways of modifying a bending-active structure so that it matches a design target: changing the cross-section of the bending elements and changing their total length. The change of cross-section of a rod changes the radius to which it will bend according to the EI value of the section. Rods of variable cross-section in bending-active shell structures could open up the variety of shapes that can be achieved, for example leading to elastic gridshells with increased usable space thanks to more vertical shapes near the ground. The ICD/ITKE bending-active structure is such an example (Fleischmann & Menges, 2012). We apply this strategy to design the arc lamps of Section 4.2.

Changing the length of the bending elements is more appropriate for structures like elastic gridshells: in this case, the local "pulling and pushing" of the rods can change the global geometry, similar to what changing the boundary geometry would do. As mentioned earlier, Garg et al. (2014) implement a similar boundary modification strategy, albeit for plastically formed gridshells rather than elastic ones. Examples of this strategy for elastic gridshells have been published by Bouhaya et al. (2014), and Soriano et al. (2019).

4.2 Varying cross-section elastica: the arc lamp

4.2.1 Methodology

First, we implement a design workflow, in Rhino/Grasshopper, where a target curve is drawn by the designer, and a closely approximating elastica of varying cross-section is found. The elastica is clamped at the base, with a weight suspended at its other end, so that it realizes an arc suitable for a lamp such as the ones depicted in Figure 4.1. This shape was selected so that it could be built by participants in a 2-day workshop. A simple example of the possible variations in this setup is presented in Figure 4.2, at a height of around 50 cm. The final designs were 1 to 1.5 m high.

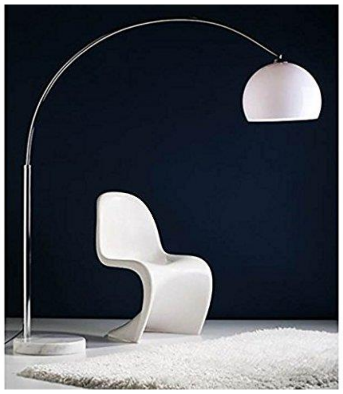
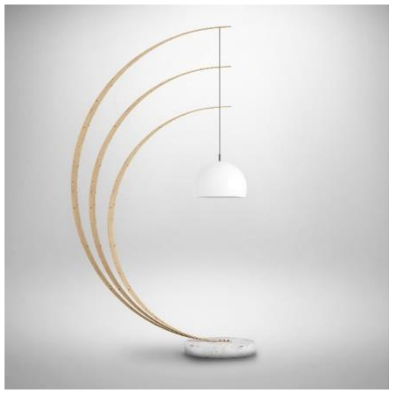
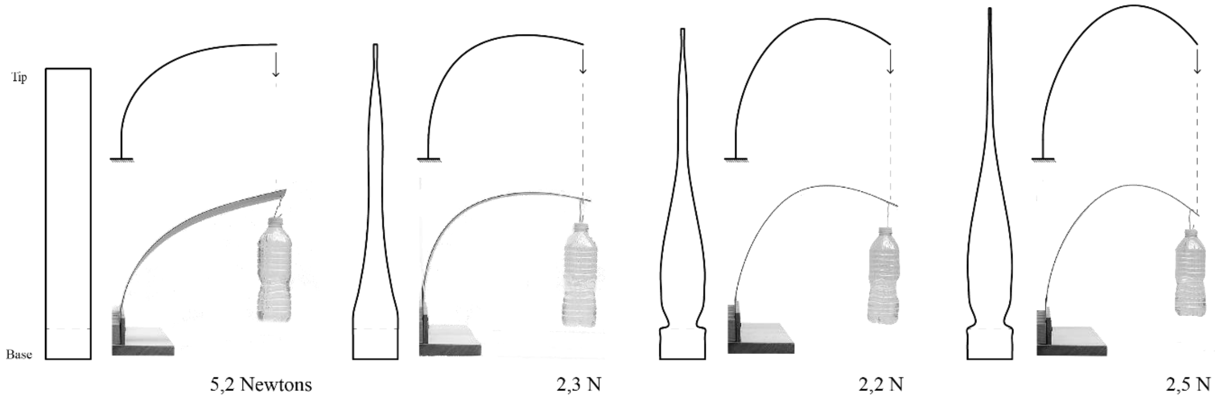



Figure 4.1: An arc lamp made of pre-bent metallic channels (left). A possible realization of an arc lamp made from initially straight timber slats (right). By carefully varying the width of the slats, the geometry can be controlled to create perfect circles.

The design workflow is based on the direct form-finding solver of Chapter 3, and a simple COBYLA optimizer for the outer loop. The elastica is discretized into 15 bending elements, and its equilibrium position is found using L-BFGS with a stopping criterion on the relative change in energy of 10^{-7}, a constraints tolerance of 10^{-4}. For the outer optimization loop, the objective is to minimize the distance from the nodes of the simulation in their equilibrium position x_i to their closest projection $P_{\mathcal{C}}(x_i)$ on the target curve, using the bending stiffness I_i of the elements of the simulation as variables:

$$(P_{4.1}) : \min_{I_i} \sum_i \|x_i - P_{\mathcal{C}}(x_i)\|^2 \quad \text{s.t.} \begin{cases} x_i = \underset{x_i}{\operatorname{argmin}}\, E_{bending}(x_i) \\ \forall i, I_{min} < I_i < I_{max} \end{cases} . \tag{4.1}$$

We restrict the bending stiffness to values that are reasonably achieved in our construction method.



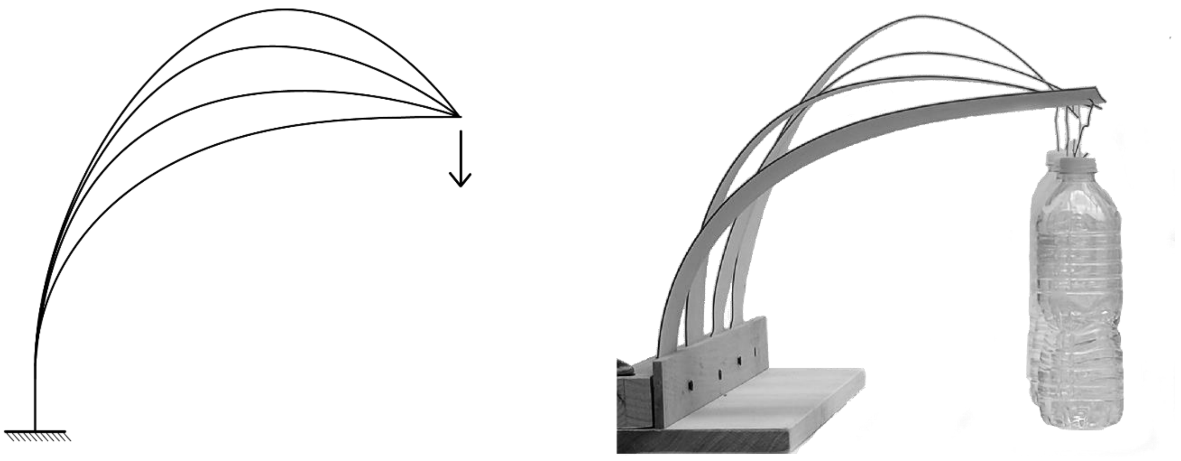


Figure 4.2: Actively bent wood strips with variable cross-section. The optimization was set up to satisfy the same boundary conditions at the base and at the location of the weight. By changing the section of the strips, the elastically bent pieces take different deformed shapes.

An initial guess on the bending stiffnesses is made using the target curve: if the solution follows it perfectly, then the bending forces at a point of the arc will equilibrate the moment generated by the point load at the end of the beam:

$$\kappa EI = FL \tag{4.2}$$

Where κ is the curvature of the target curve at that point, E the Young's modulus of the material, I the bending stiffness at that point, F the load and L the moment arm of that load to the point where the equilibrium is considered. This is not sufficient to find the solution to our inverse form-finding problem however, as this is only a local condition that gives no guarantees on the global shape. If any part of the beam cannot satisfy this condition, for example because of restrictions on the cross-section size, the rest of the elastica will quickly deviate from the target curve. In this case, a better solution will be found using our design workflow.

The interface of this design tool is shown in Figure 4.3.

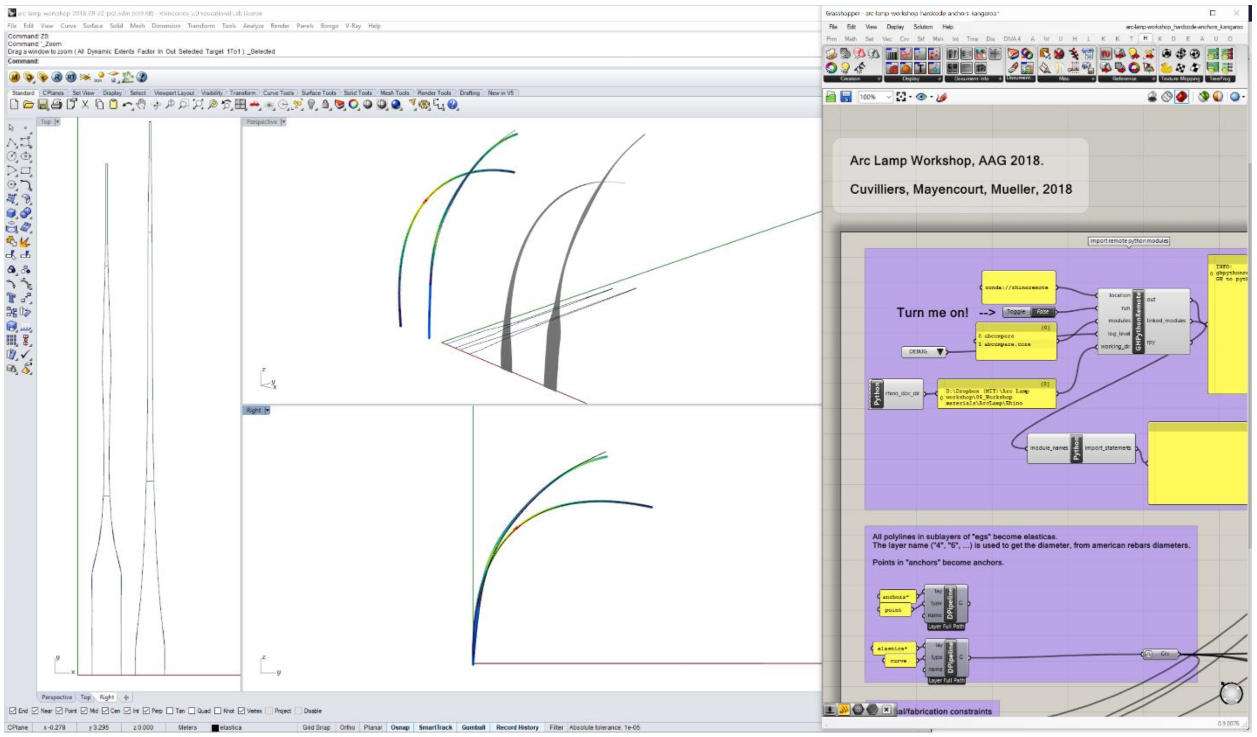

Arc Lamp Workshop, AAG 2018.
Cuvilliers, Mayencourt, Mueller, 2018
import remote python modules
Turn me on! -->
Toggle False
conda://rhinoremote
0 abcompare
1 abcompare.core
DEBUG
location
run
modules
log_level
working_dir
GHPythonRemote
out
linked_modules
rpy
rhino_doc_dir
D:\Dropbox (MIT)\Arc Lamp workshop\05_Workshop materials\ArcLamp\Rhino
module_names
import_statements
All polylines in sublayers of "egs" become elasticas.
The layer name ("4", "6", ...) is used to get the diameter, from american rebars diameters.
Points in "anchors" become anchors.
anchors*
point
elastica*
curve
DPipeline
Layer Full Path
Crv
al/fabrication constraints

Figure 4.3: Visualization of the interface of the arc lamp form-finding tool.

4.2.2 Results

The workflow was used by 7 participants of a design workshop for the Advances in Architectural Geometry conference 2018, at Chalmers University, Gothenburg, Sweden (Cuvilliers et al., 2018). Each participant designed and built an arc lamp made from 2-to-4-meter strips of furniture-grade birch plywood, from 4-mm- and 6-mm-thick sheets. The lamps were up to 2 meters tall, and 40 cm wide. The plywood was laser-cut to the shape output from the design tool, to vary the cross-sectional inertia. Additionally, multiple layers of plywood could be glued together to create stiffer sections. Figure 4.4 shows some examples of the lamps that were built and their construction details. The lamps were then put on display in the conference space, see Figure 4.5.

The workshop demonstrated that our design tool was sufficiently robust and quick to be used in an interactive design fashion. One forward simulation normally takes less than 1 second to run, with the complete result of the inverse design problem obtained in 10 to 20 seconds. This allows for quick modifications of the input target shape as a response to the previous result. See Figure 4.6 for all the lamps that were built.

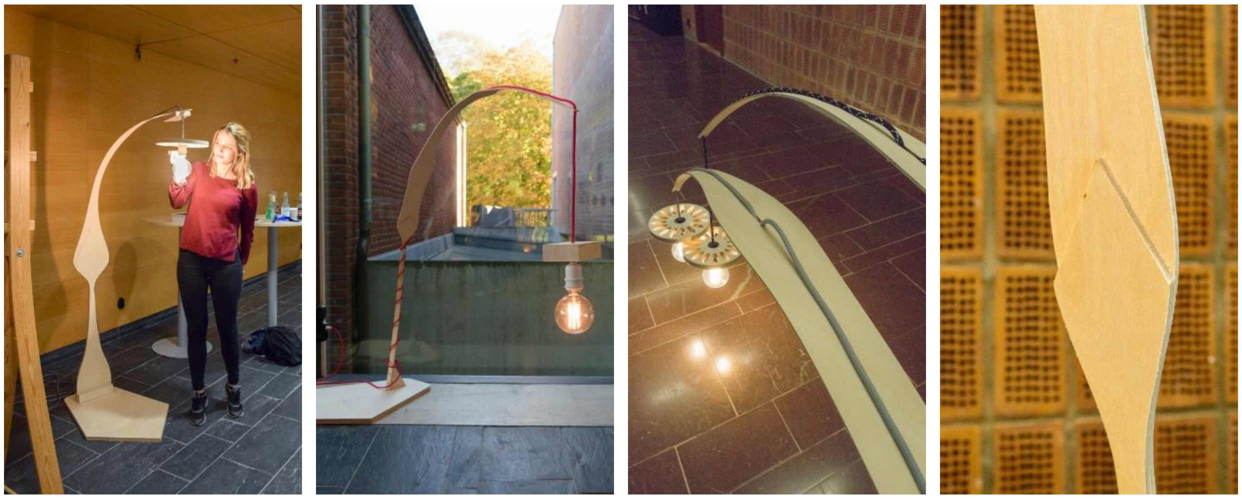


Figure 4.4: Arc lamps and their construction details.

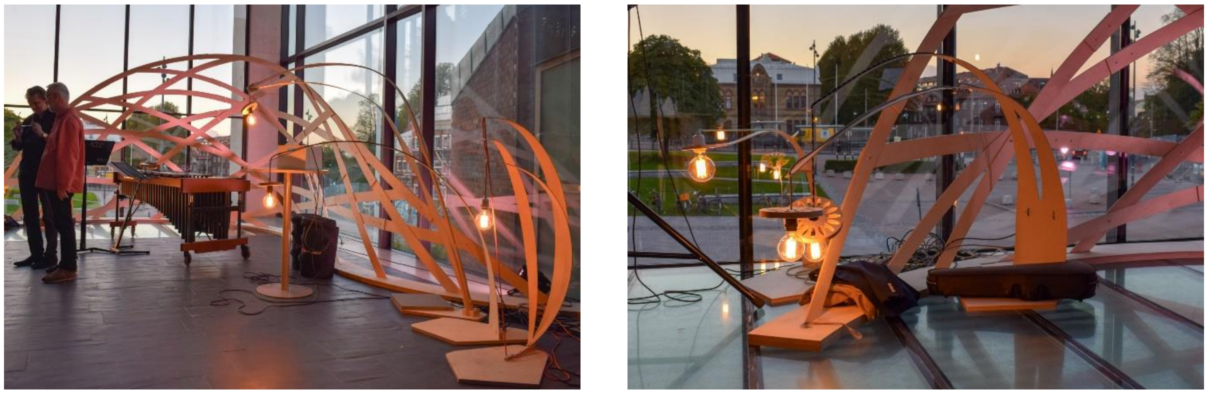


Figure 4.5: Some of the arc lamps on display in the conference space. The pre-stressed geodesic gridshell in the background was designed by Sehlström et al. (Sehlström et al., 2018).

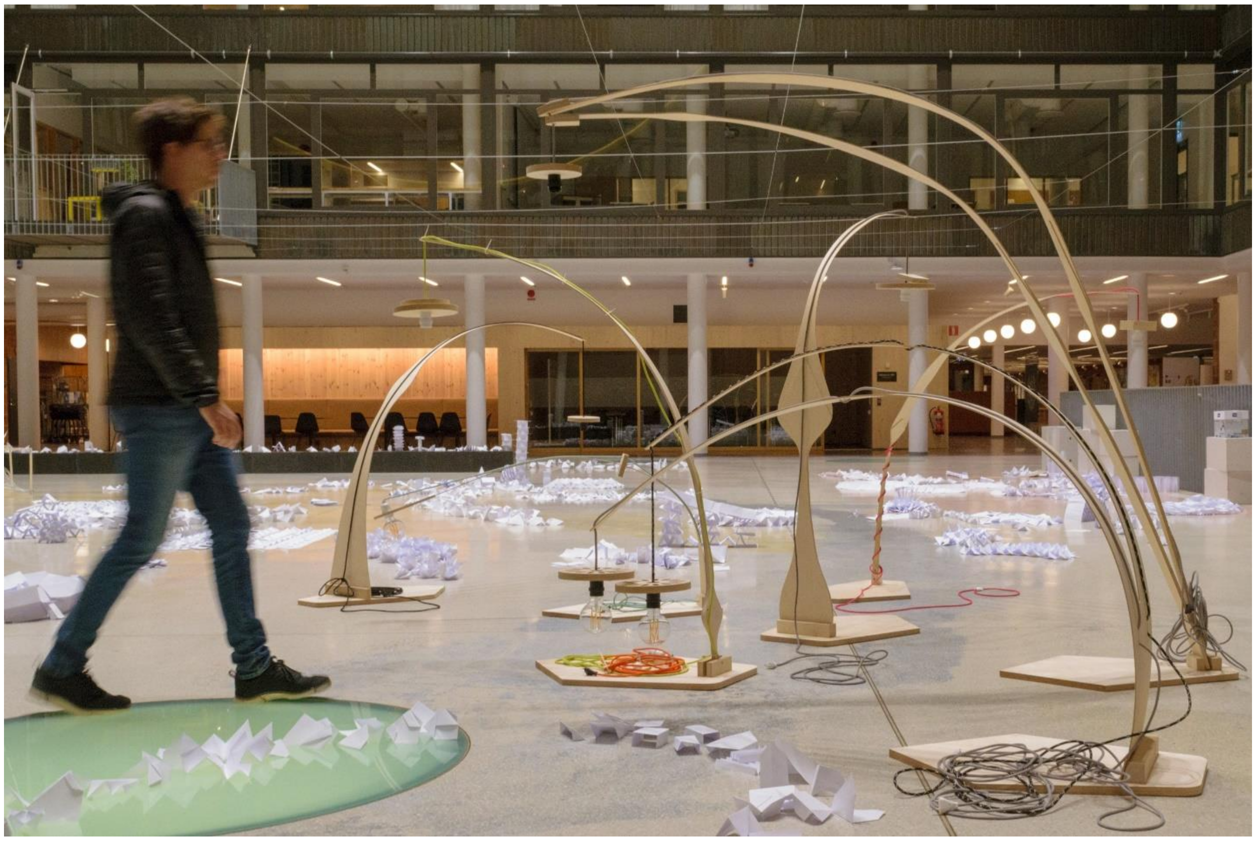


Figure 4.6: The arc lamps built for the workshop.

In pathological cases, when the target curve was far from being realizable as a bending-active equilibrium, the forward simulation often took longer to run, and we had to resort to an early stopping criterion in these cases. This was particularly the case when the target curve was slightly too straight around its base, and the initial guess slightly too stiff for the allowable width and height of the arc. Then, the forward simulation would initially be close to an equilibrium, but slowly deflect slightly more than planned around the base, which would lead to an added moment arm as the load would be further away in horizontal distance, leading to a larger deflection in the base until the structure collapsed.

Additionally, the tool lets the designer choose how the bending stiffness should be created in the construction: by increasing the width or the height of the beam. This leads to interesting shape designs of the arc front elevation, without changing its profile (as guided by the target curve). Given the limits that we had put on the width and height of the arcs, and their rate of variation, manual modifications of this would also sometimes allow for a better fit of the target curve. The interactive nature of our design tool lets the designers find these better fits easily.

Although our tool was initially made for the design of a single elastica, some workshop participants were able to modify it to model a small number of loosely coupled curves. For example, one could design two target curves with the same end points, and a shared load, so that the final construction would seem to be made of two diverging then converging arcs supporting only one lamp. Or by designing two target curves with only the same base, and playing with the maximum width of the result, two arc lamps can merge at their base.

4.3 Inverse form-finding of an elastic gridshell

4.3.1 Methodology

Next, we look at solving an inverse form-finding problem for elastic gridshells. We consider only regular grid gridshells, that have their shape formed by their boundary conditions: most laths of the gridshell will be pinned to a ground connection. We keep the same inner optimizer as previously, given the good results we obtained for a single elastica: each lath of the gridshell is discretized into at least 15 bending elements, and its equilibrium position is found using L-BFGS with a stopping criterion on the relative change in energy of 10^{-7}, a constraints tolerance of 10^{-4}. There is always at least one simulation node between two intersections on a lath, and when there are more than 7 intersections on one lath, we discretize it with 2 elements between each intersection.

For the outer optimization loop, the objective is to minimize the distance from the nodes of the simulation in their equilibrium position x_i to their closest projection $P_S(x_i)$ on the target surface. The variables are the displacements a_j of the pin anchors to the ground, from their initial positions. A similar choice of variables would be to change the lengths of the end segments on each lath that is pinned, in effect pushing and pulling the gridshell up or down from its boundary. The mathematical formulation of our optimization problem is:

$$(P_{4.2}) \colon \min_{l_i} \sum_i \|x_i - P_S(x_i)\|^2$$
$$\text{s.t.} \begin{cases} x_i = \operatorname*{argmin}_{x_i} E_{bending}(x_i) \\ \forall j, 0.1 < \dfrac{|a_j|}{l_{grid}} < 3 \end{cases}. \tag{4.3}$$

The initial guess is found by draping a regular grid over the target surface. We restrict the displacement vectors lengths $|a_i|$ between 0.1 and 3 times the grid spacing. We found bounding the variables in this way helped the optimization by preventing large variations between outer loop iterations, in turn providing faster inner solves. For the optimization, the gridshell is drawn with extra cells on its pinned boundary, so that all anchors in effect sit below the ground. This allows the anchors to be moved freely without lifting parts of the boundary above the ground. The final resulted is presented with these extra elements cut at ground level, and anchored at that intersection.

Lastly, we found that changing the variables independently could lead to ill-conditioned iterations. As we are not providing gradient information to the outer optimizer, it will approximate it with finite differencing, initially changing each variable by a fixed amount while keeping the others constant. This forces very small iterations in the beginning, as larger ones would create low-quality gridshells with very different laths lengths between consecutive laths, a situation that is either slowly solved by the inner optimizer or even completely physically unstable.

As a result, we chose instead to replace the independent variables by a crude Fourier series-like representation of them, based on the grid spacing l_{grid} of the gridshell and their initial position s_j on the boundary curve:

$$a_j = u_0 + \sum_k \left(u_k \cos\left(2\pi \frac{s_j}{\omega_k l_{grid}}\right) + v_k \sin\left(2\pi \frac{s_j}{\omega_k l_{grid}}\right)\right). \qquad (4.4)$$

When the boundary is made up of multiple disconnected segments, we repeat this process as many times as needed. See Figure 4.7 for an example of the movement of the anchors with 3 frequencies.

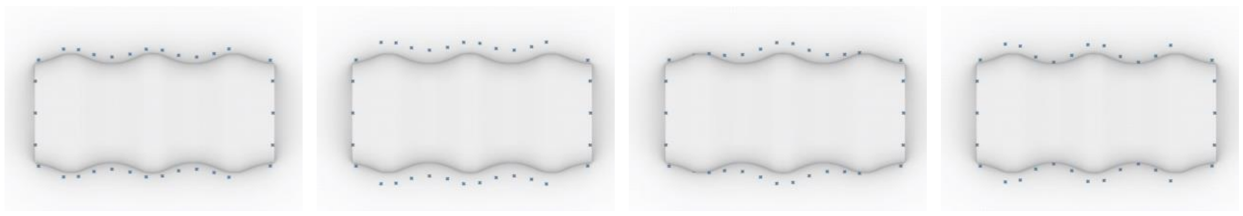


Figure 4.7: Movement of the anchors when varying coefficients for 3 frequencies. From left to right: initial positions, maximum movement with a constant offset, then with the first and second frequencies.

The new variables in the optimization problem are then the a_j and b_j amplitudes; the wavelengths ω_j are fixed at the beginning of the optimization. By changing the wavelengths ω_j between 1 and $L_{boundary}/l_{grid}$, we can generate different spatial frequencies for the features of the gridshell. Higher and lower frequencies cannot be reproduced by the grid or boundary so they can be ignored. When there are as many wavelengths as independent variables a_j, we can reproduce any combinations of a_j by changing the amplitudes u_k and v_k. However, we found that we obtained a good solution to the inverse form-finding problem (as is, close to the solution we would obtain with the original formulation with the a_j variables) by selecting only 3 to 5 frequencies that would match the main spatial frequencies of the target surface. This effectively reduces the dimensionality of the outer optimization problem without significantly reducing its design space. Lastly, by limiting the wavelengths to larger than 4, we effectively suppress the issues we described above due to the quick variations of lengths between successive laths.

Note that in this new parametrization, only the anchors are moved. The nodes of the grid keep the rules of an elastic gridshell: pinned connections at regular distances, thus preventing sliding of the bars over each other and only allowing a hinging motion at the node. The bars are initially straight, which together with the constant spacing of the nodes ensures that the gridshell can be built from an initially planar grid.

4.3.2 Results

We implemented and tested this workflow in the Rhino/Grasshopper platform, using our own implementation from Chapter 3 for the inner loop and the Goat Grasshopper plugin interface to the NLOpt optimization package for the outer optimizer, selecting the L-BFGS method. We then tested it on several structures representative of real elastic gridshells. The results are compiled in Table 4.1, where the adimensional score is calculated as:

$$\frac{\sqrt{\sum_i \|x_i - P_S(x_i)\|^2}}{n l_{grid}}, \tag{4.5}$$

that is the square root of the objective value divided by the number of intersections in the grid and the grid size. This gives us a metric that can be compared across experiments.

We focus on two main design examples: a reproduction of the Japanese Pavilion at Expo 2000 gridshell (Ban, 2003), and a parabolic dome shape that is indented at the top, summarized in Table 4.1. On these examples with up to 500 bending elements, a forward simulation can take up to .5 seconds, and the inverse form-finding solution takes 3 to 5 minutes. This allows for incremental design changes, but not true interactivity, at this scale. These structures represent two typical behaviors observed for inverse form-finding of elastic gridshells.

Table 4.1: Summary of results for our inverse form-finding framework for elastic gridshells.

	Laths [#]	Simulation nodes [#]	Simulation elements [#]	Runtime [min]	Adimensional score
Japan Pavilion	34	159	276	4.3	0.52 %
Dented dome	38	261	522	5.2	1.1 %
Valleyed dome	38	261	522	3.7	0.58 %

While the Japanese Pavilion shape lends itself very well to being built with an elastic gridshell (see Figure 4.9), the dented dome simply does not have a faithful representation with an elastic gridshell. In this case, the designer would have to edit the target shape to obtain a good solution. Figure 4.8 shows one possible modification that could be made to the dented dome target shape to find a shape that can be built with an elastic gridshell. The better agreement between the inverse form-finding result and the target shape is reflected by the improved adimensional score, see Table 4.1.

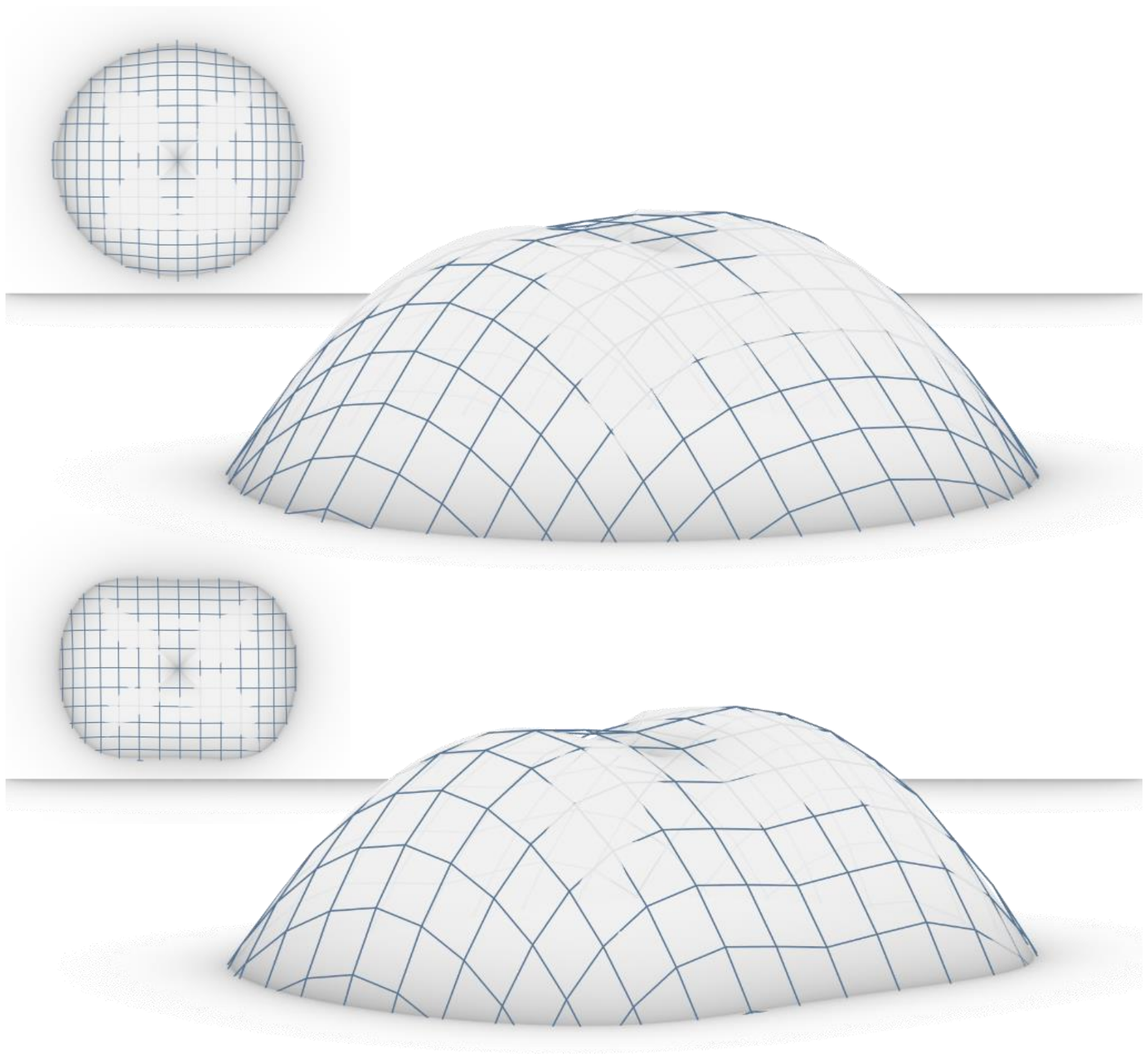


Figure 4.8: Possible modification to the dented dome target shape that can be represented by an elastic gridshell. Top: Initial shape and inverse form-finding result, with a poor agreement at the top; bottom: by continuing the dent down the sides of the target surface in a valley shape, a better fitting gridshell is found.

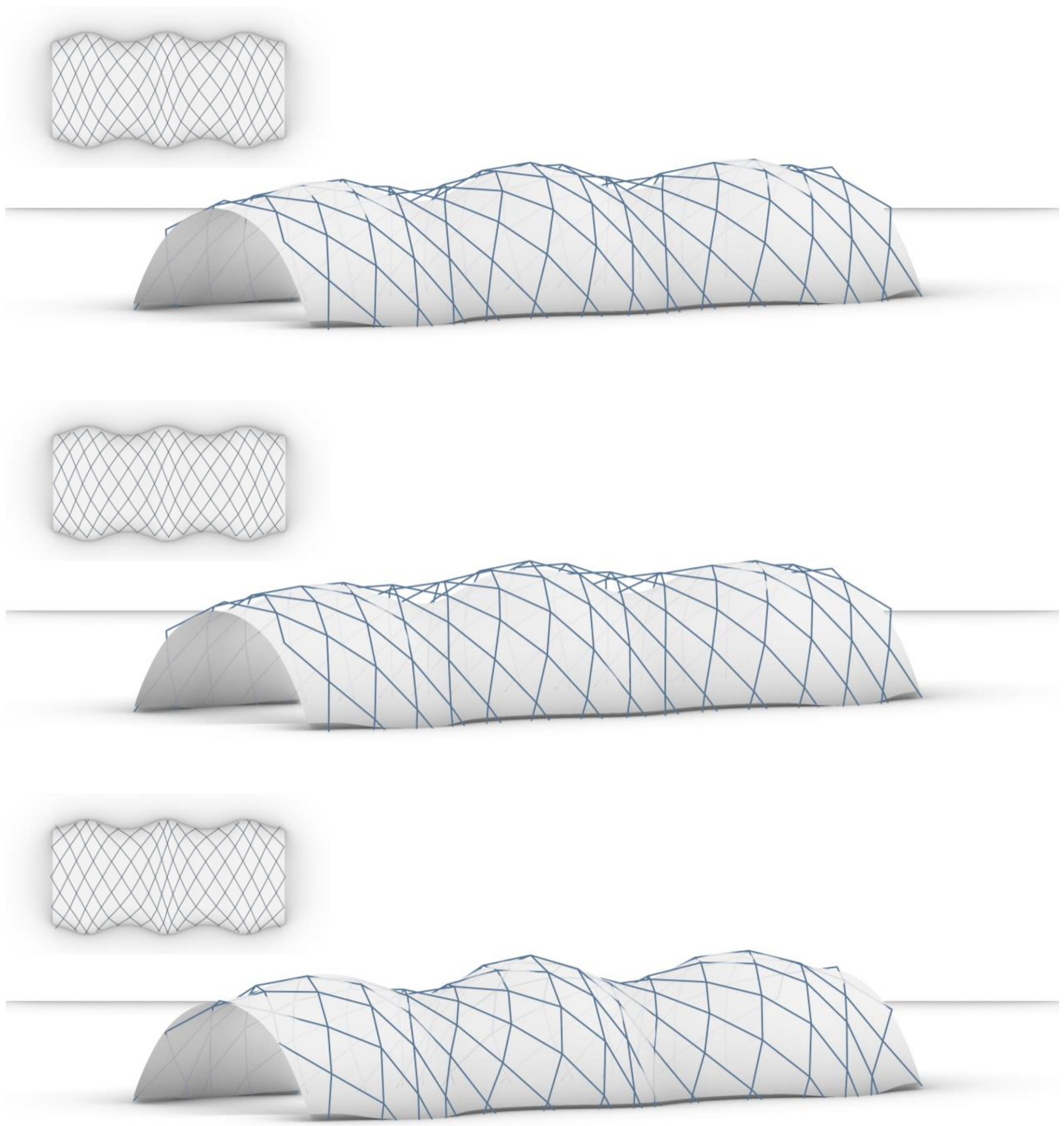


Figure 4.9: Results found by the inverse form-finding tool for elastic gridshells. (Top) Initial grid position from the draping step, not in bending equilibrium; (middle) equilibrium position of the initial grid position; (bottom) equilibrium position with optimized anchor positions resulting from the inverse form-finding tool.

4.4 Conclusion

This chapter presents a framework for the inverse form-finding of two subsets of bending-active structures: a simple elastica and regular grid elastic gridshells. The workflow has a broad range of applicability and speeds that allow for incremental modifications to the design. This lets the designer tap into a wide design space that is not typically explored for “manual” elastic gridshell designs, i.e. without inverse form-finding tools. This is in large part possible thanks to the careful selection of the inner optimizer loop presented in Chapter 3. Specifically, the speed of the inner loop directly improves the speed of the whole workflow, and its reliability lets the outer optimizer explore large design variations without fear of producing an inaccurate result that would seriously slow down the optimization process, or lead it into false minima. The Fourier-like representation of the design variables also proved very useful in speeding up and stabilizing the workflow, without reducing the quality of the results.

Several improvements could be made to this workflow to make it more useful for designers. Most importantly, the outer loop could probably be significantly sped up (thanks to many fewer inner loop calls needed) by providing gradient information to replace the finite differencing done in this work. This is for example done in (Panetta et al., 2019) for the related case of X-shells, using an adjoint method formulation. They report around one order of magnitude of speed improvement on the outer loop, with similar inner loop speeds. Additionally, the inner loop makes some simplifications on the physical construction details of elastic gridshells that could influence the results. For example, the fact that laths are not coplanar at intersections, but stacked on top one another; accounting for this leads to additional moments in the laths due to the offset forces applied through the joints. We also do not consider torsion forces in the rods.

Future work on the workflow itself could include generating better initial guesses. While the draping method has proved to work well in our case, there are other heuristics used for elastic gridshell, like the compass method (Grafe et al., 1974) to generate shear-only deformations of a regular square grid on a given surface. Additionally, there might be better ways of approximating the inverse form-finding result by incorporating some bending equilibrium information in the initial guess generation.

The initial topology of the grid largely affects the resulting forms that can be obtained, and it would be interesting to find ways to automatically explore that influence. For example in the Japan pavilion example of Figure 4.9, if the grid is oriented in a way that continuous rods cross over two ridges or two valleys, it will be very difficult to find a position of the anchors producing an equilibrium shape close to the target. One good solution might be to systematically explore how the results of the inverse form-finding change when the general orientation of the grid and the average hinge angle between the two grid directions are varied.

Similarly, our workflow does not tell the designer what could be changed in the target shape so the results would match it better. While manually generating variations of the target shape is possible as we have shown, there is no guarantee that these modifications will lead to a better result. The problem for the designer is that a trivial modification of the target shape, making it match the current inverse form-finding result, would lead to finding a perfect result but usually lose the features that they were hoping to reproduce. One way around this limitation would be to encode these target features in the objective function, and use convergence information to decide which ones are impossible to match by the optimization. This is an approach we are investigating in Chapter 6.

Elastic gridshells are a powerful example of quickly-deployable bending-active structures, that can span large spaces while remaining lightweight. By implementing inverse form-finding workflows for these, we allow designers to realize a larger panel of shapes with them, and have better control on modifying these shapes. There are still large classes of similar problems where similar workflows could be beneficial, both for deployable structures and bending-active structures. The methods developed in this work, and the integration of inverse form-finding techniques, could similarly expand the available design spaces of hard-to-design structural systems where only forward simulations are currently available.

5 Inverse form-finding of funicular structures: target shape

5.1 Introduction

Compression-only structures are much more efficient than structures where bending occurs. For example, in a linear structural element, slender by definition, axial loads are straight forces resisted uniformly by the whole section, while bending is a force with a large moment arm resisted by the section where only small moment arms exist. Thus, by focusing on funicular structures for a given load, we guarantee that all the material will be used with the most efficiency. This explains the significant interest in such structures historically and today.

The compression-only state in a structure is similar to a perfectly flexible chain with no self-weight, fixed at both ends with some slack, hanging under the action of weights attached to it. The chain undergoes tension only, and if its shape were inverted, it would resist the same loads in a compression-only state. This principle is at the root of many physical experiments for the form-finding of funicular structures, such as Antoni Gaudí's (1852-1926) hanging models of the Colonia Guell church (Huerta, 2006).

In general, designing compression-only structures is challenging. Physical experiments of hanging chains provide a good way to interact and quickly iterate on design options but lack the precision of a CAD model. Computationally, finding the shape of a set of hanging chains requires accounting for nonlinear behavior due to large displacements; this problem can be alleviated using the force density method (Schek, 1974). However, this only solves a *direct problem*: finding one discrete funicular shape under given loads and grid properties. Often, this goes against the intuitive design process where the designer has a shape in mind and wants to minimize bending. That describes an *inverse problem* where the closest possible funicular structure to a target surface is found.

This study aims to solve one such inverse problem: construct funicular structures as close as possible to a target surface. The scope is limited to grid-like, node-and-branch only networks, for which an efficient calculation procedure exists in large displacements. In the particular case of grid-like funicular structures, this inverse problem can be solved using a genetic algorithm (Block & Lachauer, 2011). This has the advantage of accepting even badly formatted problems, but remains slow and lacks a guarantee of finding a global optimum. Van Mele and Block (2011) present a more formal treatment of the problem, but the scope is limited to pre-tensioned cable nets, for which a good initial estimate of the solution is known. This can be overcome using thrust network analysis (Panozzo et al., 2013), but leads to a slower multi-step optimization process. This chapter expands on this previous work by using gradient-based optimization methods to gain more insight into the solutions for such closest-fit inverse problems.

5.2 Background

5.2.1 Funicular bar networks

We consider networks of nodes connected by bars, with free rotations at the nodes. This guarantees that there are only constant axial forces in the bars. The equilibrium at each node is then only a consequence of the position of the nodes. Then for a network of bars intersecting at the nodal positions $\boldsymbol{x}$, under the loads $\boldsymbol{p}$ at the nodes and with the objective surface S, the problem has the form:

$$(P_{5.1}): \min_{\mathrm{x}} d(\boldsymbol{x}, S)^2 \text{, such that at all nodes: } \sum \boldsymbol{F} = \boldsymbol{p}, \qquad (5.1)$$

where d is a distance measuring the fitness of the points on the surface. This is a convex objective function with nonlinear constraints. This is the general closest-fit problem formulation for funicular bar networks; the specifics of the distance function d used in this research are given in Section 5.3.

5.2.2 Force density method

The force density method (Schek, 1974), is a method well-suited to explore a large number of funicular structures resulting from the same initial bar network. Three assumptions are made: (i) every bar is elastically stretched proportionally to the force it carries following Hooke's law (constitutive equation), (ii) the length of a bar is equal to the distance between the nodes that it connects (compatibility equation) and (iii) each node is in equilibrium (equilibrium equation). Mathematically, this becomes:

$$\begin{cases} \boldsymbol{C_N}^T\boldsymbol{Q}\boldsymbol{C_N}\boldsymbol{x_N} + \boldsymbol{C_N}^T\boldsymbol{Q}\boldsymbol{C_F}\boldsymbol{x_F} - \boldsymbol{p_x} = 0 \\ \boldsymbol{C_N}^T\boldsymbol{Q}\boldsymbol{C_N}\boldsymbol{y_N} + \boldsymbol{C_N}^T\boldsymbol{Q}\boldsymbol{C_F}\boldsymbol{y_F} - \boldsymbol{p_y} = 0 \\ \boldsymbol{C_N}^T\boldsymbol{Q}\boldsymbol{C_N}\boldsymbol{z_N} + \boldsymbol{C_N}^T\boldsymbol{Q}\boldsymbol{C_F}\boldsymbol{z_F} - \boldsymbol{p_z} = 0 \end{cases}. \tag{5.2}$$

Here, $\boldsymbol{C_N}$ is the edge matrix for the free nodes, taking value -1 for the start node of a bar and +1 at its end node; $\boldsymbol{C_F}$ is the edge matrix for the fixed nodes; $\boldsymbol{x}$, $\boldsymbol{y}$, and $\boldsymbol{z}$ are the positions of the nodes; $\boldsymbol{Q}$ is the diagonal matrix of the force densities; and $\boldsymbol{p}$ is the vector of all external loads. Recall that the force density q is the ratio of the force in a bar, s, to its length, l:

$$q = s/l. \tag{5.3}$$

We set for clarity $\boldsymbol{D_N} = \boldsymbol{C_N}^T\boldsymbol{Q}\boldsymbol{C_N}$ and $\boldsymbol{D_F} = \boldsymbol{C_N}^T\boldsymbol{Q}\boldsymbol{C_F}$. The nodal positions solving a direct problem can then be found using only linear algebra:

$$\begin{cases} \boldsymbol{x_N} = \boldsymbol{D_N}^{-1}(\boldsymbol{p_x} - \boldsymbol{D_F}\boldsymbol{x_F}) \\ \boldsymbol{y_N} = \boldsymbol{D_N}^{-1}(\boldsymbol{p_y} - \boldsymbol{D_F}\boldsymbol{y_F}) \\ \boldsymbol{z_N} = \boldsymbol{D_N}^{-1}(\boldsymbol{p_z} - \boldsymbol{D_F}\boldsymbol{z_F}) \end{cases}. \tag{5.4}$$

However, finding the solution to an inverse problem, where the target shape is known and the force densities are the unknown, is not evident, as there is no clear link between the two.

5.2.3 Rationalization of freeform surfaces with funicularity constraints

As shown above, different choices of force densities naturally lead to different funicular solutions. This is a consequence of the linearization of the system of equilibrium equations where, physically, different stiffness is assigned to each bar of the network. If we recall that the initial problem is to solve the equilibrium of a bar network, it is cogent that different distributions of stiffness yield different equilibrium shapes. We can use these force densities to obtain a funicular network that fits a target surface as close as possible. The strategy is illustrated in Figure 1.1 and formalized in Section 5.3.

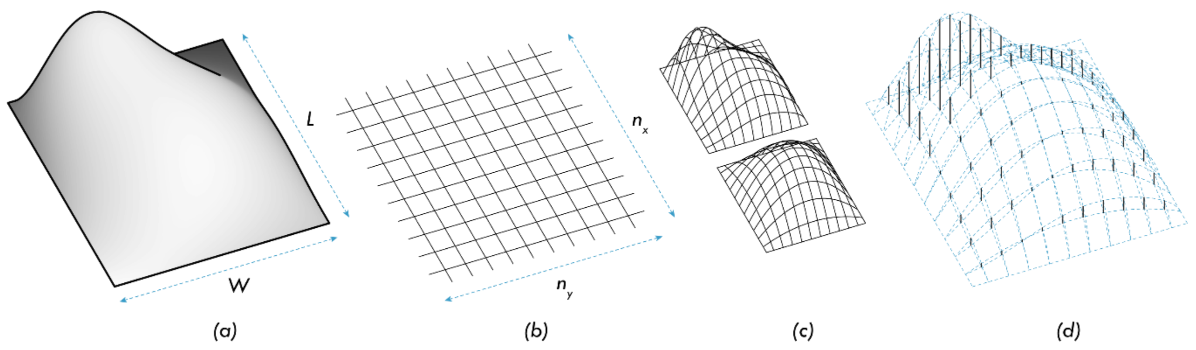


Figure 5.1: Rationalizing a freeform surface (a): (b) choose numbers of branches n_x and n_y in the x and y directions, (c) approximate the target surface by projecting the grid vertically and (d) compare to the funicular network found using the force density method. Then, optimize the force densities to reduce the distance.

5.3 Problem Formulation

To keep the problem tractable, we constrain it from the outset by imposing that the nodes will only move vertically from their initial positions. This can be achieved by reducing the number of free parameters, i.e. by imposing that each bar of a branch has the same force density. This constraint is derived by analyzing the horizontal equilibrium of a single node in a quadrilateral grid. In particular, this means that two bars initially aligned must carry the same horizontal force, related to their force density and length by Equation (5.3). Given that we restrict our problem to a regular rectangular grid, the lengths of two bars belonging to the same branch must thus be equal and so must be their force densities. The matrix $\boldsymbol{K}$, whose entries are all 0 or 1, is introduced to link the bar force densities q_i's to the $n_x + n_y$ independent branch force densities:

$$\boldsymbol{q} = \boldsymbol{K}\boldsymbol{b}. \tag{5.5}$$

Since we have restricted our problem to vertical displacements only, we are able to devise a simple metric for the distance between a bar network and our target surface. This metric is defined as follows:

$$d(\boldsymbol{z}_N) = \|\boldsymbol{z}_N - \boldsymbol{z}_T\|^2\,, \tag{5.6}$$

where $\boldsymbol{z}_N$ is the vector of the z-coordinates of the free nodes of the funicular bar network and $\boldsymbol{z}_T$ is the vector of the target z-coordinates, i.e. the z-coordinates of the grid nodes projected on the target surface.

5.3.1 Unconstrained Problem

The inverse form-finding problem is the nonlinear, unconstrained optimization problem formulated as follows:

$$(P_{5.2}): \min_{\boldsymbol{b}} F(\boldsymbol{b}) = \left\| \left(\boldsymbol{D}_N{}^{-1}(\boldsymbol{p}_z - \boldsymbol{D}_F \boldsymbol{z}_F) \right) - \boldsymbol{z}_T \right\|^2, \tag{5.7}$$

where $\boldsymbol{D}_N = \boldsymbol{C}_N{}^T diag(\boldsymbol{q})\boldsymbol{C}_N$, $\boldsymbol{D}_F = \boldsymbol{C}_N{}^T diag(\boldsymbol{q})\boldsymbol{C}_F$, and $\boldsymbol{q} = \boldsymbol{K}\boldsymbol{b}$. Even though $(P_{5.2})$ is not convex for all values of $\boldsymbol{b}$, convexity can be found and gradient-based methods for solving it are still available if the problem is constrained to a smaller domain.

5.3.1.1 Gradient

To find the gradient of the objective function, we first look for the Jacobian $\boldsymbol{J}(\boldsymbol{b})$ of the function $\boldsymbol{f}(\boldsymbol{b}) = \boldsymbol{z}_N(\boldsymbol{b}) - \boldsymbol{z}_T$. Using the chain rule, we get:

$$\boldsymbol{J}(\boldsymbol{b}) = \frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}_N} \frac{\partial \boldsymbol{z}_N(\boldsymbol{q})}{\partial \boldsymbol{q}} \frac{\partial \boldsymbol{q}(\boldsymbol{b})}{\partial \boldsymbol{b}}. \tag{5.8}$$

With $\frac{\partial \boldsymbol{f}}{\partial \boldsymbol{z}_N} = 1$, $\frac{\partial \boldsymbol{z}_N(\boldsymbol{q})}{\partial \boldsymbol{q}} = -\boldsymbol{D}_N{}^{-1}\boldsymbol{C}_N{}^T\boldsymbol{W}$ (from (Schek, 1974)), and $\frac{\partial \boldsymbol{q}(\boldsymbol{b})}{\partial \boldsymbol{b}} = \boldsymbol{K}$, we get: $\boldsymbol{J}(\boldsymbol{b}) = -\boldsymbol{D}_N{}^{-1}\boldsymbol{C}_N{}^T\boldsymbol{W}$. $\boldsymbol{W}$ is the diagonal matrix of the lengths of the bars projected on the z axis: $\boldsymbol{W} = diag(\boldsymbol{C}_N\boldsymbol{z}_N + \boldsymbol{C}_F\boldsymbol{z}_F)$. Finally, the gradient of the objective function is:

$$\nabla_{\boldsymbol{b}} \|\boldsymbol{z}_N - \boldsymbol{z}_T\|^2 = -2\, \boldsymbol{K}^T.\boldsymbol{W}.\boldsymbol{C}_N.\boldsymbol{D}_N{}^{-\mathrm{T}}.(\boldsymbol{z}_N - \boldsymbol{z}_T). \tag{5.9}$$

5.3.1.2 Hessian

It is also possible to obtain the Hessian of $F(\boldsymbol{b})$ in a similar fashion. This derivation, or a similar one for equivalent problems, was not found in the literature by the author.

$$\begin{aligned} \boldsymbol{H}(\boldsymbol{b}) &= \frac{\partial^2 \|\boldsymbol{z} - \boldsymbol{z}_T\|}{\partial \boldsymbol{b}^2} = 2 * \frac{\partial}{\partial \boldsymbol{b}} \left((\boldsymbol{z} - \boldsymbol{z}_T)^T . \frac{\partial (\boldsymbol{z} - \boldsymbol{z}_T)}{\partial \boldsymbol{b}} \right) \\ &= 2 * \left(\boldsymbol{J}(\boldsymbol{b})^T . \boldsymbol{J}(\boldsymbol{b}) + \sum_i (\boldsymbol{z} - \boldsymbol{z}_T)_i . \frac{\partial^2 z_i(\boldsymbol{b})}{\partial \boldsymbol{b}^2} \right) \end{aligned} \tag{5.10}$$

We used an explicit summation to lift any ambiguity on the third-order tensor contraction. To get $\frac{\partial^2 \boldsymbol{z}_i(\boldsymbol{b})}{\partial \boldsymbol{b}^2}$, we apply the chain rule twice on $\boldsymbol{b} = \boldsymbol{b}(\boldsymbol{q})$ and note that $\boldsymbol{K}$ is constant in $\boldsymbol{b}$, to reuse the expression of $\boldsymbol{J}(\boldsymbol{b})$:

$$\begin{aligned} \frac{\partial^2 z_i(\boldsymbol{b})}{\partial \boldsymbol{b}^2} &= \frac{\partial}{\partial \boldsymbol{b}} \left(\frac{\partial z_i(\boldsymbol{b})}{\partial \boldsymbol{q}} . \frac{\partial \boldsymbol{q}}{\partial \boldsymbol{b}} \right) = \frac{\partial}{\partial \boldsymbol{b}} \left(\frac{\partial z_i(\boldsymbol{b})}{\partial \boldsymbol{q}} . \boldsymbol{K} \right) = \frac{\partial^2 z_i(\boldsymbol{b})}{\partial \boldsymbol{b} \partial \boldsymbol{q}} . \boldsymbol{K} = \boldsymbol{K}^T . \frac{\partial^2 z_i(\boldsymbol{b})}{\partial \boldsymbol{q}^2} . \boldsymbol{K} \\ &= \boldsymbol{K}^T . \frac{\partial}{\partial \boldsymbol{q}} (-\boldsymbol{D}^{-1}\boldsymbol{C}^T\boldsymbol{W})_{i,.} . \boldsymbol{K} , \end{aligned} \tag{5.11}$$

where $(-\boldsymbol{D}^{-1}\boldsymbol{C}^T\boldsymbol{W})_{i,.}$ is the *i*th row of $\boldsymbol{J}(\boldsymbol{b})$. Then, for each component, using equation (59) from Petersen and Pedersen (2012) for the one variable derivative of the inverse of a matrix:

$$\left\{\frac{\partial^2 z_\alpha(\boldsymbol{b})}{\partial \boldsymbol{q}^2}\right\}_{i,j} = \left\{\frac{\partial}{\partial q_i}(-\boldsymbol{D}^{-1}\boldsymbol{C}^T\boldsymbol{W})_{\alpha,j}\right\}_{i,j} = \left\{\left[\boldsymbol{D}^{-1}\boldsymbol{C}^T\frac{\partial \boldsymbol{Q}}{\partial q_i}\boldsymbol{C}\boldsymbol{D}^{-1}\boldsymbol{C}^T\boldsymbol{W}\right]_{\alpha,j}\right\}_{i,j}. \quad (5.12)$$

Since $\boldsymbol{Q} = \text{diag}(\boldsymbol{q})$, $\frac{\partial \boldsymbol{Q}}{\partial q_i}$ is a matrix of zeros with only one 1 on the diagonal in row i. Then, we have $\boldsymbol{C}^T\frac{\partial \boldsymbol{Q}}{\partial q_i}\boldsymbol{C} = C_{i,i}^2 = 1$ since $\boldsymbol{C}$ is made entirely of 1 and -1. We get:

$$\left\{\frac{\partial^2 z_\alpha(\boldsymbol{b})}{\partial \boldsymbol{q}^2}\right\}_{i,j} = \left\{[\boldsymbol{D}^{-2}\boldsymbol{C}^T\boldsymbol{W}]_{\alpha,j}\right\}_{i,j} = \begin{pmatrix}[\boldsymbol{D}^{-2}\boldsymbol{C}^T\boldsymbol{W}]_{\alpha,1}\\ \vdots \\ [\boldsymbol{D}^{-2}\boldsymbol{C}^T\boldsymbol{W}]_{\alpha,n_q}\end{pmatrix}_{i,j}. \quad (5.13)$$

Finally by replacing $\frac{\partial^2 z_i(\boldsymbol{b})}{\partial \boldsymbol{b}^2}$ with its value, reorganizing the sum and compacting it to a matrix product:

$$\boldsymbol{H}(\boldsymbol{b}) = 2 * \left(\boldsymbol{J}(\boldsymbol{b})^T\boldsymbol{J}(\boldsymbol{b}) + \boldsymbol{K}\begin{pmatrix}(\boldsymbol{z}-\boldsymbol{z}_T).[\boldsymbol{D}^{-2}\boldsymbol{C}^T\boldsymbol{W}]\\ \vdots \\ (\boldsymbol{z}-\boldsymbol{z}_T).[\boldsymbol{D}^{-2}\boldsymbol{C}^T\boldsymbol{W}]\end{pmatrix}\boldsymbol{K}^T\right). \quad (5.14)$$

This matrix has a rank deficiency of 1, the consequences of which are discussed in the example problem of Section 5.4.1.1 and in general in Section 5.3.3.

5.3.2 Structure of the unconstrained problem

To gain insight into the problem, the simplest example comprised of four bars and one node is analyzed. Although the bar network has four bars, it only has two branches, hence two independent force parameters, which will allow us to visualize the objective function. The physical problem is presented visually in Figure 5.2. The goal of the optimization is to have $\boldsymbol{z}_N$ reach $\boldsymbol{z}_T$ by modifying the independent force densities of the network, namely b_1 and b_2. In this particular problem, it is obvious that it is possible to fit a funicular solution to the target since the target is itself funicular. This simple problem is thus well-suited not only for understanding the problem but also to test out algorithms. In the numerical applications, L, W, $\boldsymbol{z}_T$, and $\boldsymbol{p}_z$ are respectively equal to 10, 10, 4, and 10.

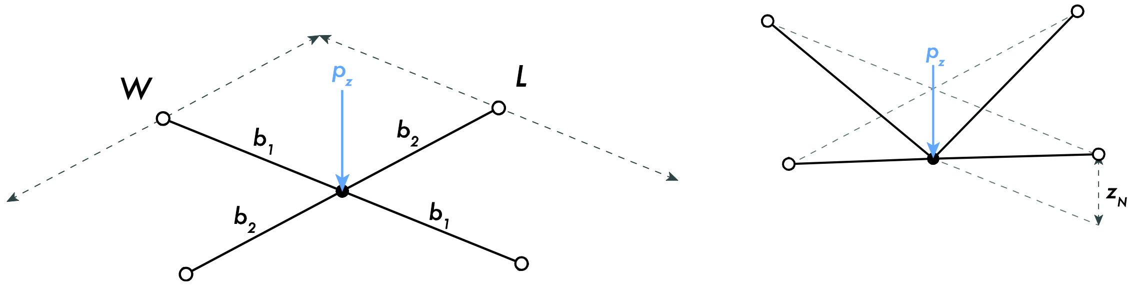


Figure 5.2: The four-bar problem.

As seen in Figure 5.3 (left), the problem is nonconvex and has a line of global minima. The level sets of the objective function are lines with the following form:

$$F_\alpha(b_1, b_2) = \begin{cases} b_1 + b_2 = C_\alpha \\ z = \alpha \end{cases}, \tag{5.15}$$

where C_α is a constant depending on α. From the form of the level sets, we see that the gradient always has the same direction $(1,1)$; only its scale and sign will change. However, depending on the starting point, a computational optimization scheme based on the gradient will not necessarily converge. This is particularly clear when looking at the section of the objective surface by the plane $b_1 = b_2$. For a starting point (b_0^1, b_0^2), a gradient-based algorithm will converge only if at all steps:

$$(b_k^1, b_k^2) \in \{(b_1, b_2) \in \mathbb{R}^2 \mid b_1 + b_2 > 0\}. \tag{5.16}$$

A partial solution to the non-convexity issue is to reformulate our problem and constrain it to the positive orthant. This will solve the problem of the choice of the starting point. Moreover, in the four-bar example, the plane $b_1+b_2 = 0$ is particular because it corresponds to a state of physical instability: with these force densities, the bars cannot equilibrate the vertical force. In general, this is a problem that can arise if we allow for force densities of different signs. By restricting the problem to non-negative densities, the issue is simply avoided.

5.3.3 Structure of the constrained problem

We constrain the base set of problem $(P_{5.2})$ to the non-negative orthant, to obtain $(P_{5.3})$:

$$(P_{5.3})\colon \min_b \; F(\boldsymbol{b}) = \left\| \left(\boldsymbol{D}_N{}^{-1}(\boldsymbol{p}_z - \boldsymbol{D}_F \boldsymbol{z}_F) \right) - \boldsymbol{z}_T \right\|^2 \quad \textbf{s.t. } \boldsymbol{b} \geq \boldsymbol{0} \tag{5.17}$$

Two details are of importance in the structure of $(P_{5.3})$. First, the objective shape is in general not a funicular shape so the optimum value will be an unknown positive number. This means that solvers cannot be stopped based on the current function value being closed to zero, but only based on the improvement in the objective function or the size step.

Second, as mentioned in Section 5.3.1.2, the Hessian matrix of $(P_{5.2})$ has a rank deficiency of 1. The same goes for the matrices $\boldsymbol{C}$ and $\boldsymbol{A} = \mathrm{abs}(\boldsymbol{C})$ of the absolute values of the components of $\boldsymbol{C}$. $\boldsymbol{A}$ represents the indices of the bars connected to each node, or equivalently the indices of the forces acting on each node, and so is locally representative of the structure of the possible equilibrium positions. Reordering the rows of $\boldsymbol{C}$, the structure of $\boldsymbol{A}$ can be written:

$$\boldsymbol{A} = \left(\begin{array}{c|ccccc} & \multicolumn{5}{c}{\overbrace{\hphantom{1\quad 0\quad 0\quad \dots\quad 0}}^{n_x}} \\ & 1 & 0 & 0 & \dots & 0 \\ \boldsymbol{I}_{n_y} & \vdots & \vdots & \vdots & \dots & \vdots \\ & 1 & 0 & 0 & \dots & 0 \\ & 0 & 1 & 0 & \dots & 0 \\ \boldsymbol{I}_{n_y} & \vdots & \vdots & \vdots & \dots & \vdots \\ & 0 & 1 & 0 & \dots & 0 \\ \vdots & & & \vdots & & \\ & 0 & 0 & \dots & 0 & 1 \\ \boldsymbol{I}_{n_y} & \vdots & \vdots & \dots & \vdots & \vdots \\ & 0 & 0 & \dots & 0 & 1 \end{array}\right), \tag{5.18}$$

with n_x and n_y the number of independent force density values in the x and y directions, respectively, and supposing $n_x < n_y$. This matrix has a rank deficiency of 1, giving an indeterminacy in the force density method problem: a linear combination of the independent force densities will give the same result in the final shape. Given the structure of A, one could believe that those equivalent values are in a vector space of dimension 1; however, the positions of the nodes at equilibrium are related to $\boldsymbol{A}$ and $\boldsymbol{b}$ only after multiplications by the lengths of the bars to get to forces acting on the nodes. There is no clear structure to the set of equivalent independent force densities values.

This indeterminacy is detrimental to the quality of the optimization processes. At every point one direction of the problem will always be flat, so the solver will never search in that direction even though it might be on a shorter path to the optimum. Also, it means that solutions exist with unbalanced values of the independent force densities. This is also generally unwanted physically – it leads to a high concentration of forces in certain bars.

One possible remedy is to remove the indeterminacy by adding the constraint $\sum_{i=1}^{nx} b_i = \sum_{i=1}^{nx} b_i$, following the structure of the 4-bar problem. While this works well in the unconstrained problem, it stops the algorithm too soon in the constrained problem. In fact, when the algorithm finds an optimum on the frontier of one of the inequality constraints and the equality constraint proposed above, it will not be able to get away from the barrier in the flat direction at that point to then follow a descent direction again. This constraint was not used in the rest of this chapter except for part of Section 5.4.2.

5.4 Results

5.4.1 Steepest Descent Algorithm (SDA)

In this section, we apply the steepest descent Algorithm 5.1 to the unconstrained problem. The gradient of the objective function with respect to the independent force densities was given by Equation (5.9). The gradient can be used to provide a descent direction; as a matter of fact, it provides the steepest descent direction. The step-size is determined by computing $\alpha^k \coloneqq \underset{\alpha}{\operatorname{argmin}} F\left(b^k + \alpha d^k\right)$. Hence, in order to find α^k, one has to solve:

$$2\left(\left(D_N{}^{-1}C_N{}^T W\right)\Big|_{b=b^k+\alpha d^k} * K * J(b^k)^T * (z_N - z_T)\big|_{b=b^k}\right)^T * (z_N - z_T)\big|_{b=b^k+\alpha d^k} = 0. \quad (5.19)$$

No closed-form solution exists in general and we must resort to an approximate line-search by bisection.

Algorithm 5.1: Steepest Descent Algorithm (SDA)

```
Initialize at $b^0$, and set $k \leftarrow 0$
At iteration $k$:
    1. $d^k \coloneqq -\nabla F\left(b^k\right) = -2\,J\left(b^k\right)^T\left(z_N^k - z_T\right)$. If $||d^k|| \leq \varepsilon$, then stop ($\varepsilon$ is specified tolerance)
    2. Choose step-size $\alpha^k$ (by performing a line-search)
    3. Set $x^{k+1} \leftarrow x^k + \alpha^k d^k, k \leftarrow k+1$
```

5.4.1.1 Example 1: 4-bar problem

We use Algorithm 5.1 on the four-bar problem introduced in Section 5.3.2. We apply the steepest descent algorithm with and without line-search. Our starting point is $\boldsymbol{b}_0 = (10{,}10)$.

With the line-search, the algorithm converges in one step, which was expected given the structure of the problem. We note however that we had to modify the bisection algorithm as to avoid overshooting past the plane $b_1 + b_2 = 0$, i.e. we had to set an upper bound for the choice of the step-size. Because we knew the objective surface beforehand and knowingly chose an easy starting point, this constraint was easy to implement as follows:

$$\alpha^k \leq \min\left(\left|\frac{b^k}{d^k}\right|\right), \tag{5.20}$$

where b^k/d^k indicates a component-wise division. Generally, this constraint is not necessarily as easy to formulate. This problem is a motivation for constraining the problem to non-negative force densities. Without the line-search (arbitrary step-size of 0.001), the algorithm expectedly converges very slowly as seen in Figure 5.3 (right).

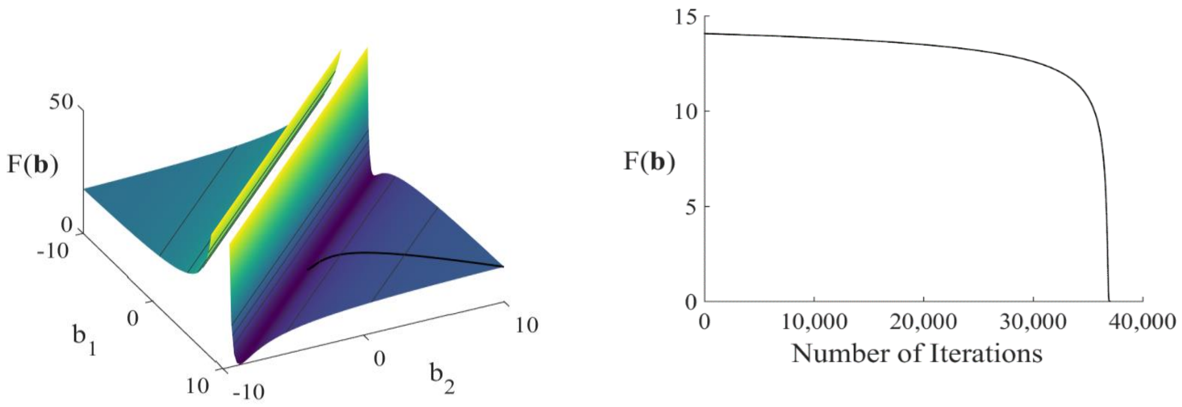


Figure 5.3: (left) The objective surface with the optimization path (black curve) for the steepest descent algorithm with a fixed step-size of 0.001. At every iteration, the direction of the gradient remains unchanged because the level sets of the function are parallel lines. (right) Convergence profile of the steepest descent algorithm with a step-size of 0.001, for the 4-bar problem.

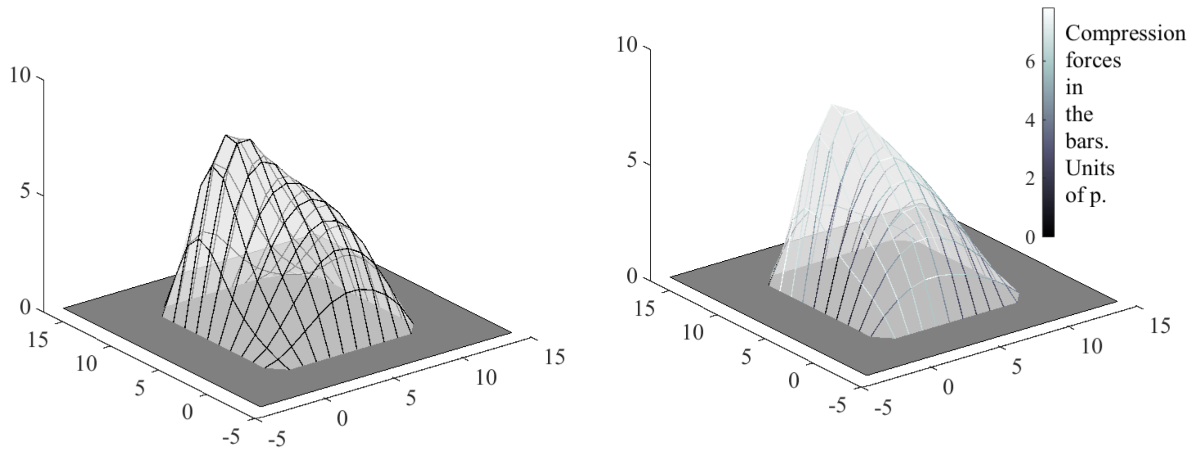


Figure 5.4: Target surface (left) vs. optimum surface found (right).

5.4.1.2 Example 2: 10x10-bar problem

In this example, we deal with a larger dimension problem, i.e. a grid network of size 10 by 10 nodes. The target surface and optimum found are presented in Figure 5.4. Applying the steepest descent algorithm with the starting point $\boldsymbol{b}^0 = [10 \ldots 10]^T$, where $\boldsymbol{b}^0$ has 20 components, the problem converges to the optimum solution ($F^* = 38.1539$) in 4213 iterations. Convergence is defined by a relative change in the objective value of less than 10^{-6}. Figure 5.5 (a) shows that the algorithm converges quickly in the first 20 iterations but considerably slows downs afterward. If instead, we use a starting point $\boldsymbol{b}^0 = [1 \ldots 1]^T$, the algorithm does not converge. A solution is to set up an arbitrarily low step-size, as in the previous example but it results in an exceptionally slow algorithm. Again, these observations motivate us to switch to a constrained problem.

5.4.2 Quasi-Newton Methods

In this section, we focus on finding a performant algorithm for solving $(\boldsymbol{P_{5.3}})$ in the 100 nodes examples of section 5.4.1.2. Looking at the poor performance of the SDA in a realistic problem, we implemented a Newton Method (NM) with line-search, Algorithm 5.2. Because the method relies on inverting the Hessian, non-invertible as per Section 5.3.3, it was necessary to enforce the additional constraint $\sum_{i=1}^{nx} \boldsymbol{b_i} = \sum_{i=1}^{nx} \boldsymbol{b_i}$. The constraint is written as an additional line in the Hessian and the gradient is augmented by one component equal to $\mathbf{0}$ to get a well-formed system of equations in the first step of Algorithm 5.2.

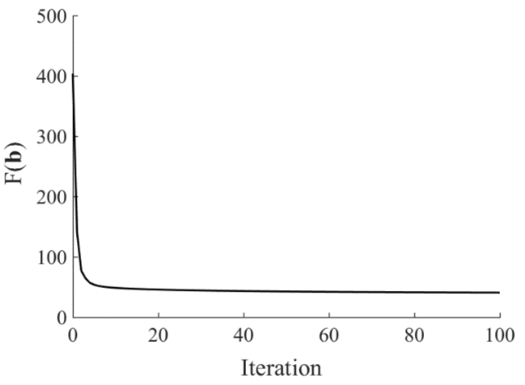
Algorithm 5.2: Newton's method (NM)

Initialize at b^0, and set $k \leftarrow 0$. Let $\varepsilon > 0$ be a given error tolerance.

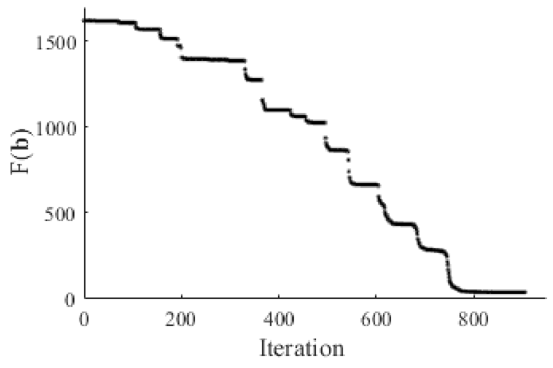
At iteration k:

1. Set $H_{const}(b^k) = \begin{pmatrix} H(b^k) \\ \underbrace{1 \quad \cdots \quad 1}_{nx} \quad \underbrace{-1 \quad \cdots \quad -1}_{ny} \end{pmatrix}$, $\nabla F_{const}(b^k) = \begin{pmatrix} \nabla F(b^k) \\ 0 \end{pmatrix}$.
2. $d^k := -H_{const}(b^k)^{-1} \nabla F_{const}(b^k)$. If $\|d^k\| \leq \varepsilon$, then stop.
3. Choose step-size $\alpha^k = \operatorname{argmin}_{\alpha \geq 0} F(b^k + \alpha d^k)$.
4. Set $x^{k+1} = x^k + \alpha^k d^k, k = k + 1$

However, given the narrow zone in which the objective function of $(P_{5.3})$ is convex, this method only converges with a starting point very close to the optimum. In practice, we were only able to obtain convergence by using a starting point found as a result of the SDA, stopped when the relative change in the objective value was 10^{-3}.



(a)



(b)

Figure 5.5: Convergence profile of (a) the steepest descent algorithm with an inexact line-search, and (b) of the interior-point method for the 10x10 bar problem.

5.4.2.1 Levenberg-Marquardt algorithm

To have a better convergence rate for the final iterations along with a large convergence domain, we used the Levenberg-Marquardt algorithm (Levenberg, 1944; Marquardt, 1963) implementation in MATLAB (The Mathworks Inc., 2015) for non-linear least-square problems. This is the method used by Van Mele et al. (2011).

The Levenberg-Marquardt algorithm uses a search direction that is intermediate between the one found in the SDA and the one found in NM. A parameter is used to orient the amount by which the problem is similar to the SDA or the NM. At the beginning, this parameter is chosen so that the problem is very close to the SDA (to get a large convergence region) and is progressively updated to match more closely the NM (to get a faster convergence rate). Because the problem is never exactly the NM, the non-invertible Hessian is not a problem anymore and we can eliminate the additional constraint used above.

Using this method, we were able to find a solution with the same optimum value as in Section 5.4.1.2, in 1585 iterations using the same stopping criterion. However with several values of the force densities found are negative, which is unacceptable if we were to build the solution. In consequence, we look at constrained solvers to find solutions to $(P_{5.3})$ in both reasonable time and large convergence.

5.4.3 Solvers for the constrained problem

Using the solvers implemented in MATLAB's *fmincon* package, we looked at their relative performance for solving $(P_{5.3})$ in the 100 nodes example presented in Section 5.4.1.2. The results are compiled in Table 5.1. The solvers are always stopped when the relative improvement in objective value is less than 10^{-6}.

Table 5.1: Performance comparison of constrained nonlinear solvers in the 100 nodes example.
(*An additional constraint was added to the interior point method.)

Algorithm	**Iterations**	**Function calls**	**Objective value**
Interior point*	887	1070	38.2510
Sequential Quadrating Programming	923	1431	38.2356
Trust region reflective	28	29	38.2356

The gradient is given explicitly to the solvers. The Hessian is computed numerically by the solvers, as some of them do not accept a user-defined analytical expression for the Hessian.

Convergence was always obtained when the initial point was a vector of identical independent force densities within one order of magnitude (above or below) of the uniform load at the nodes. It should be noted that the excellent convergence rate of the Trust region reflective algorithm fades when the stopping criterion is made smaller.

For the interior point method, the solver quickly gets stuck against a bound and, because it relies heavily on the Hessian, does not explore in the flat direction of the problem. When the additional constraint $\sum_{i=1}^{nx} b_i = \sum_{i=1}^{nx} b_i$ is added, we get the convergence as for the other algorithms, this is the result presented in Table 5.1. This explains the different objective values found at the optimum. See Figure 5.5 (b) for a typical evolution of the objective values using the interior-point method from a poor starting point.

5.5 Conclusions

In this chapter, we analyzed the structure of a closest-fit inverse form-finding problem for funicular structure. Using the force density method framework, we optimized the force densities of a bar network to reach a target surface. We formulated the problem first as an unconstrained optimization problem and calculated the analytic expressions of the gradient and Hessian of the problem. Based on this, we applied the steepest descent algorithm on a 4-bar problem. The insight gained from this analysis motivated the introduction of a non-negative force densities constraint. The constraint was dealt with computationally by using nonlinear constrained optimization methods that proved effective, especially the interior point and trust-region reflective methods. We note that, for particular surfaces, the constraint will yield solutions that are sub-optimal compared to solutions of the unconstrained problem. Yet, both for physical and for computational stability motives, the constraint introduced is useful.

Future work should be focused on handling realistic construction constraints, for example limits on the forces in the bars or smaller deviations of those forces. Looking at self-weight funicular structures, where the applied load is a function of the geometry of the shell, is also promising. Finally, the optimization strategy presented in this work could be integrated into a user-guided design tool used by designers to directly create and interact with funicular structures.

6 Inverse form-finding of funicular structures: Functional design objectives

This chapter builds on and improves the simple inverse form-finding system for funicular surfaces of Chapter 5 to produce a feature-based representation of the design objective. Thanks to a functional representation of this objective, combined with automatic differentiation and a "perfect" forward simulation process – that is fast and always produces a result, we construct a workflow that not only does inverse form-finding towards a target shape, but to any objective that the designer can represent as a function of the properties of the current iteration. This is a novel way of looking at inverse form-finding design, that we find very promising in its potential uses.

6.1 Introduction

In our exploration of inverse form-finding, we have so far focused on a single type of objective: a target shape. However, not all design intents can be efficiently encoded as a target shape. For example, they cannot represent design objectives such as "aim at having all members of the structure be the same length". We hinted at this issue in Chapter 5, and before that in Chapter 2 when we looked at structures built of several construction systems that each need to be form-found. While many of these issues have been tackled separately, as part of a forward form-finding system usually, we propose in this chapter a different approach and instead aim at extending the outer optimizer of an inverse form-finding system so that it can optimize for multiple and diverse objective types (combining several objectives in a composite objective function if needed).

While many general-purpose optimizers can accommodate this, we have found that to obtain meaningful results in a limited time, they work best when combined with a technique called Automatic Differentiation that could provide exact and fast gradient values to the optimizer at each iteration. To be able to optimize for general user-coded objectives, and because most general-purpose optimizers require some gradient information to converge, we need to have a way to derive that information automatically. Otherwise, we would have to resort to a catalog of pre-coded objectives that the designer could select, and manually compute and code their gradients. The traditional way of solving this issue is to use finite differencing, a technique we used in Chapter 4. However, this is a slow process, especially when the number of variables is large as it requires on the order of one forward simulation per variable to compute the gradient. Thus, we constructed our inverse form-finding framework for funicular structures so that it could use automatic differentiation.

Automatic differentiation, a computational tool that automatically finds an exact yet efficient gradient of general procedures, has recently seen renewed interest due to its usefulness in machine learning algorithms (Baydin et al., 2018). The tools developed there are useful for many other optimization problems, such as those encountered in form-finding problems with constraints or objectives of best-fit to target. In this work, we will demonstrate this by building a differentiable framework for the form-finding of funicular surfaces, with additional constraints. The word differentiability here is taken to mean that the system will be able to output gradients for all its numerical procedures, without the programmer explicitly encoding them. Adding an objective will only require a function giving its value, the gradient is automatically computed from this function.

Because we set up our problem as an inverse form-finding problem with nested solvers as we have done several times now in this dissertation, we have the guarantee that all results generated will solve the forward form-finding simulation. Specifically, here we use the force density method to form-find funicular shapes, this means that all the results we will extract from our complete workflow will be funicular, and optimize as best we can the objectives given by the designer. This will let us explore the design space of funicular structures in a very directed and efficient way. This has the potential to significantly change the way we usually look at inverse form-finding, as the designer will now be able to encode much more complex features than approaching a target shape. It also paves the way for multi-criteria design, where for example a funicular structure is optimized to be close to a shape buildable by an elastic gridshell, or optimized to reduce the energy consumption of the space it encloses, for example.

Generally, the optimization problems we solve here are very close to what we did in Chapter 5, but with a general objective function. Also, we directly use the force densities as variables rather than the independent ones on square grids, to be able to use any kind of meshing:

$$(P_{6.1}):\ \min_{q} f(\mathbf{x}),\ \text{where } \mathbf{x} = \mathbf{D_N}^{-1}(\mathbf{p} - \mathbf{D_f}\mathbf{x_f}). \tag{6.1}$$

For example, we could find a funicular surface that is close to a target surface and minimizes the length differences in its members by choosing the objective function as:

$$f(\mathbf{x}) = \sum_{i} \left\| x_i - x_i^{target} \right\|^2 + \sum_{j} \left(l_j - l^{target} \right)^2 . \tag{6.2}$$

6.2 Background

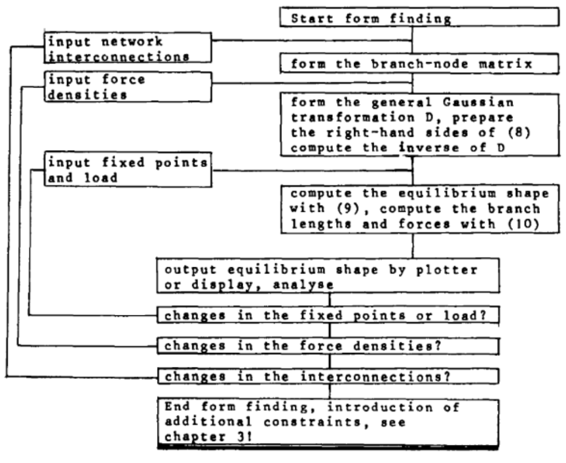
6.2.1 Related work

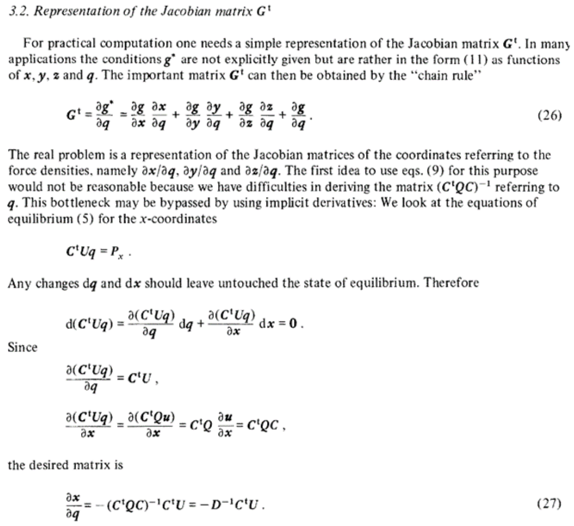
Our approach is based on the force density method as expressed for the first time by Schek (1974). This lets us generate equilibrium shapes for meshes, given their constitutive properties of connectivity (connectivity matrix $\mathbf{C}$) and stiffness (force densities $\mathbf{Q}$), and external loads $\mathbf{p}$, that is funicular shapes (see Chapter 5 for more details):

$$\begin{cases} \boldsymbol{x}_N = \boldsymbol{D}_N{}^{-1}(\boldsymbol{p}_x - \boldsymbol{D}_F\boldsymbol{x}_F) \\ \boldsymbol{y}_N = \boldsymbol{D}_N{}^{-1}(\boldsymbol{p}_y - \boldsymbol{D}_F\boldsymbol{y}_F) \\ \boldsymbol{z}_N = \boldsymbol{D}_N{}^{-1}(\boldsymbol{p}_z - \boldsymbol{D}_F\boldsymbol{z}_F) \end{cases},\ \text{where } \boldsymbol{D}_N = \boldsymbol{C}_N{}^T\boldsymbol{Q}\boldsymbol{C}_N \text{ and } \boldsymbol{D}_F = \boldsymbol{C}_N{}^T\boldsymbol{Q}\boldsymbol{C}_F. \tag{5.4}$$

We use these shapes as the basis of a design space and then optimize for various objectives inside this design space.

This is related to other design systems for funicular structures that use thrust network analysis, a method related to the force density method, to generate funicular forms, and optimize these to match a best-fit-to-target objective (Block & Lachauer, 2011; Van Mele et al., 2014). Here, we use the force density method as a generator, tack on additional objectives on our designs, and integrate this in an optimization program, with the force densities as variables. Similarly, several non-linear force density methods formulations have been proposed to introduce additional constraints to the force density method, such as to account for time-dependent behavior (Kmet & Mojdis, 2015), material non-linearities in membranes (Koohestani, 2014), or to fix some nodal positions or reaction forces and solve for compressive elements (Malerba et al., 2012; Miki & Kawaguchi, 2010). These cited works also use an optimization loop to solve their respective specific non-linear problems. Even in the original force density paper (Schek, 1974), a workflow around the optimization of the force density method's results is already described, see Figure 6.1. In these and previous systems however, there is a high cost associated with adding a new objective, because an explicit procedure computing its gradient is required to get an efficient optimization program.


Start form finding
input network interconnections
form the branch-node matrix
input force densities
form the general Gaussian transformation D, prepare the right-hand sides of (8) compute the inverse of D
input fixed points and load
compute the equilibrium shape with (9), compute the branch lengths and forces with (10)
output equilibrium shape by plotter or display, analyse
changes in the fixed points or load?
changes in the force densities?
changes in the interconnections?
End form finding, introduction of additional constraints, see chapter 3!


3.2. Representation of the Jacobian matrix $\boldsymbol{G}^{\mathrm{t}}$

For practical computation one needs a simple representation of the Jacobian matrix $\boldsymbol{G}^{\mathrm{t}}$. In many applications the conditions $\boldsymbol{g}^*$ are not explicitly given but are rather in the form (11) as functions of $\boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}$ and $\boldsymbol{q}$. The important matrix $\boldsymbol{G}^{\mathrm{t}}$ can then be obtained by the "chain rule"

$$\boldsymbol{G}^{\mathrm{t}} = \frac{\partial \boldsymbol{g}^*}{\partial \boldsymbol{q}} = \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{x}} \frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{y}} \frac{\partial \boldsymbol{y}}{\partial \boldsymbol{q}} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{z}} \frac{\partial \boldsymbol{z}}{\partial \boldsymbol{q}} + \frac{\partial \boldsymbol{g}}{\partial \boldsymbol{q}} . \qquad (26)$$

The real problem is a representation of the Jacobian matrices of the coordinates referring to the force densities, namely $\partial \boldsymbol{x}/\partial \boldsymbol{q}$, $\partial \boldsymbol{y}/\partial \boldsymbol{q}$ and $\partial \boldsymbol{z}/\partial \boldsymbol{q}$. The first idea to use eqs. (9) for this purpose would not be reasonable because we have difficulties in deriving the matrix $(\boldsymbol{C}^{\mathrm{t}}\boldsymbol{Q}\boldsymbol{C})^{-1}$ referring to $\boldsymbol{q}$. This bottleneck may be bypassed by using implicit derivatives: We look at the equations of equilibrium (5) for the x-coordinates

$$\boldsymbol{C}^{\mathrm{t}}\boldsymbol{U}\boldsymbol{q} = \boldsymbol{P}_x .$$

Any changes $\mathrm{d}\boldsymbol{q}$ and $\mathrm{d}\boldsymbol{x}$ should leave untouched the state of equilibrium. Therefore

$$\mathrm{d}(\boldsymbol{C}^{\mathrm{t}}\boldsymbol{U}\boldsymbol{q}) = \frac{\partial(\boldsymbol{C}^{\mathrm{t}}\boldsymbol{U}\boldsymbol{q})}{\partial \boldsymbol{q}} \mathrm{d}\boldsymbol{q} + \frac{\partial(\boldsymbol{C}^{\mathrm{t}}\boldsymbol{U}\boldsymbol{q})}{\partial \boldsymbol{x}} \mathrm{d}\boldsymbol{x} = \boldsymbol{0} .$$

Since

$$\frac{\partial(\boldsymbol{C}^{\mathrm{t}}\boldsymbol{U}\boldsymbol{q})}{\partial \boldsymbol{q}} = \boldsymbol{C}^{\mathrm{t}}\boldsymbol{U} ,$$

$$\frac{\partial(\boldsymbol{C}^{\mathrm{t}}\boldsymbol{U}\boldsymbol{q})}{\partial \boldsymbol{x}} = \frac{\partial(\boldsymbol{C}^{\mathrm{t}}\boldsymbol{Q}\boldsymbol{u})}{\partial \boldsymbol{x}} = \boldsymbol{C}^{\mathrm{t}}\boldsymbol{Q} \frac{\partial \boldsymbol{u}}{\partial \boldsymbol{x}} = \boldsymbol{C}^{\mathrm{t}}\boldsymbol{Q}\boldsymbol{C} ,$$

the desired matrix is

$$\frac{\partial \boldsymbol{x}}{\partial \boldsymbol{q}} = -(\boldsymbol{C}^{\mathrm{t}}\boldsymbol{Q}\boldsymbol{C})^{-1}\boldsymbol{C}^{\mathrm{t}}\boldsymbol{U} = -\boldsymbol{D}^{-1}\boldsymbol{C}^{\mathrm{t}}\boldsymbol{U} . \qquad (27)$$

Figure 6.1: The optimization system for design objectives described by Schek (1974), and some of the gradients calculations needed.

Additionally, there are existing systems that have been presented for the design of funicular surfaces, that do not use the linearization opportunities of the force density method. For example, Vouga et al. (2012) use a formulation of the equilibrium of funicular surfaces discretized with discrete differential geometry tools on meshes, and solve the resulting non-linear problems using a conjugate gradient solver to find a funicular surface close to a target surface. De Goes et al. (2013) improve on this framework both in terms of solving speed and details encoded in the material behavior's discretization. Finally, Tang et al. (2014) discretize the same constitutive equations in a subdivision surfaces framework to allow for true 3-dimensional target shapes, rather than the 2.5-dimensional heightmaps from the previous works. While all these workflows encode more details and precision in their formulation of the constitutive equations of funicular surfaces than a simple force density method formulation, this is only true for meshes that represent continuous surfaces. For rigid gridshells made of discrete linear elements, the force density method formulation that lumps the membrane behavior in the edges of the mesh is sufficient. More importantly, they are all restricted in the objectives that they can solve for, usually a target surface. The reason is that gradients of the objective function are always needed to produce a form-finding workflow with sufficient speed and stability. Because these need to be coded manually, the designer is not able to add different objectives or even combine existing ones in different ways. Using only a target surface as the objective seriously the level of abstraction that can be encoded in this objective, and prevents us from exploring the design space of funicular surfaces in more complex directions than finding the closest one to a target surface.

6.2.2 Automatic differentiation

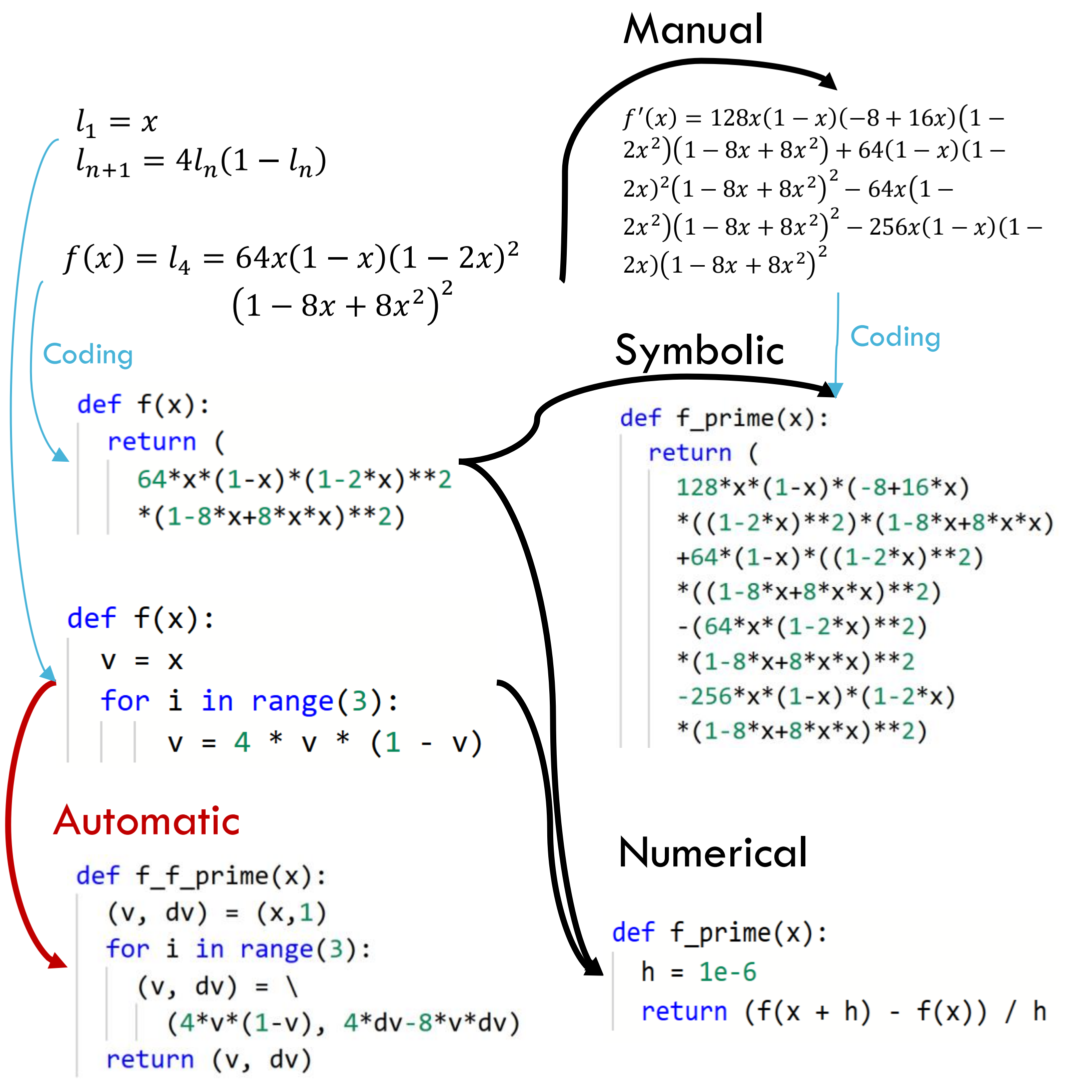
To overcome these limitations, we propose to use automatic differentiation. Automatic differentiation is a computational process in which a software function is automatically transformed to produce derivative values as well as the regular function value. It differs from usual ways of software differentiation in important ways, see Figure 6.2:

- It does not require manually deriving an expression for the gradient of the function, as would a code not using any kind of software differentiation
- It produces code that is often faster than what symbolic differentiation produces. Symbolic differentiation recognizes known mathematical expressions in the function taken as one operation, derives them, and combines and simplifies them using classical mathematical rules. Compared to automatic differentiation, it is a slow process that does not deal well with software constructs such as branches and loops.
- It produces exact values for the gradients, and does not suffer from numerical instabilities like numerical differentiation (or finite differencing) does. Finite differencing finds derivatives by approximating with the limit definition of derivatives: $f'(x) \approx \big(f(x+h) - f(x)\big)/h$, for h small.

Instead, automatic differentiation attaches to each basic mathematical operation information about how to compute its derivatives, formally replacing code that computes $f: v \longmapsto f(v)$ by $f: (v,\, \mathrm{d}v) \longmapsto (f(v),\, f'(v)\mathrm{d}v)$. It then computes and combines them as the code is executed. This is a mathematically exact operation, that also works through branches and loops. This means the calculation of derivatives will follow a very similar code path than the simple function, and so be on the same order of execution time.

There are two main modes to automatic differentiation: forward mode and reverse mode. The forward mode is the simplest in its implementation, it combines derivatives using a chain rule. Imagining that our function f can be decomposed into a series of basic mathematical operations $f = f_1 \circ f_2 \circ \ldots \circ f_n$, we have:

$$\partial f(v) = \partial f_1\big(f_2 \circ \ldots \circ f_n(v)\big) \times \partial f_2\big(f_3 \circ \ldots \circ f_n(v)\big) \times \ldots \times \partial f_n(v). \qquad (6.3)$$



$$l_1 = x$$
$$l_{n+1} = 4l_n(1 - l_n)$$

$$f(x) = l_4 = 64x(1-x)(1-2x)^2(1-8x+8x^2)^2$$

$$f'(x) = 128x(1-x)(-8+16x)(1-2x^2)(1-8x+8x^2) + 64(1-x)(1-2x)^2(1-8x+8x^2)^2 - 64x(1-2x^2)(1-8x+8x^2)^2 - 256x(1-x)(1-2x)(1-8x+8x^2)^2$$

```
def f(x):
    return (
        64*x*(1-x)*(1-2*x)**2
        *(1-8*x+8*x*x)**2)
```

```
def f(x):
    v = x
    for i in range(3):
        v = 4 * v * (1 - v)
```

```
def f_prime(x):
    return (
        128*x*(1-x)*(-8+16*x)
        *((1-2*x)**2)*(1-8*x+8*x*x)
        +64*(1-x)*((1-2*x)**2)
        *((1-8*x+8*x*x)**2)
        -(64*x*(1-2*x)**2)
        *(1-8*x+8*x*x)**2
        -256*x*(1-x)*(1-2*x)
        *(1-8*x+8*x*x)**2)
```

```
def f_f_prime(x):
    (v, dv) = (x,1)
    for i in range(3):
        (v, dv) = \
            (4*v*(1-v), 4*dv-8*v*dv)
    return (v, dv)
```

```
def f_prime(x):
    h = 1e-6
    return (f(x + h) - f(x)) / h
```

Figure 6.2: Different ways of obtaining derivatives for a computational function: manual differentiation, symbolic, numerical, and automatic. After Baydin et al. (2018).

So if, when computing $f_i(x)$, where $x = f_{i+1} \circ \ldots \circ f_n(v)$, we also compute and save $\partial f_i(x)$ thanks to the derivative function attached to each basic operation, we will be able to compute ∂f by propagating these derivatives forward through the function code. The reverse mode is similar but uses the adjoint method to propagate derivatives backward, in effect finding which variation of the input gives a given variation of the output.

These two modes mainly differ in practice when applied to multiple-valued functions of multiple variables. When applied to $f\colon \mathbb{R}^n \to \mathbb{R}^m$, forward takes on the order of n times the number of operations needed to compute f; reverse mode takes on the order of m times the number of operations needed to compute f. So in our case, where we mostly have scalar objective functions of many design variables, the reverse mode will be much more adequate.

Automatic differentiation is a relatively old technique but has remained sparsely used until recently, when the increased interest for machine learning and specifically deep learning made its characteristic extremely desirable. The first ideas were outlined by Beda et al. (1959), with the first generalist implementations appearing in the 1980s (B. Speelpenning, 1980). Barthelemy and Hall (1995) were first to use it for engineering design and optimization of complex systems integrating structures, among other things. Big developments were made with the release of modern deep learning frameworks, which all integrated some form of automatic differentiation that could be repurposed for general use: Theano (Bergstra et al., 2010), TensorFlow (Abadi et al., 2016), Autograd (Maclaurin et al., 2014/2015), PyTorch (Paszke et al., 2019). It is now being applied to a wider variety of problems in engineering where the objective function is either not possible to differentiate mathematically, or user-supplied and so unknown at coding time. For example, we can cite applications in topology optimization (Nørgaard et al., 2017), robot physical simulation for gait learning (Degrave et al., 2017), rendering and material properties learning (Wu et al., 2017), or protein structure prediction (Ingraham et al., 2019).

6.3 Methodology

We solve ($P_{6.1}$) by implementing the force density method in a package for automatic differentiation of vector and matrix computations (specifically, we use Autograd (Maclaurin et al., 2014/2015)), and attach it to a numerical solver for non-linear problems (we used SciPy, generally with L-BFGS). The resulting framework can generate funicular shapes with arbitrary objectives as defined by the designer. Because no gradients are required from the designer, it is easy to modify the output by changing the constraints or objectives. For example, we have experimented with objectives that make all elements lengths similar, or chosen from a list of possible lengths, as well as flat panels, similar angles, and best-fit-to-target. The optimization programs are efficiently solved, returning in a runtime reasonable for iterative design even for problems with as many as 10,000 elements.

We note that the force density method is a good example of a form-finding process for experimenting with an automatic differentiation workflow, in that it is a direct problem that is well-behaved, where the results of automatic differentiation can be compared to manual differentiation. It is also typical in its mathematical structure, with its sparse connectivity and many variables attached to the edges and vertices, and in its intent, as it gives one solution among many. Our workflow is then an exploration tool in the design space generated by the force density method for a fixed mesh connectivity, as the force densities are varied – they are our design variables.

The objectives can be specified as small Python functions, that depend on the equilibrium position of any intermediary result.

6.4 Case studies

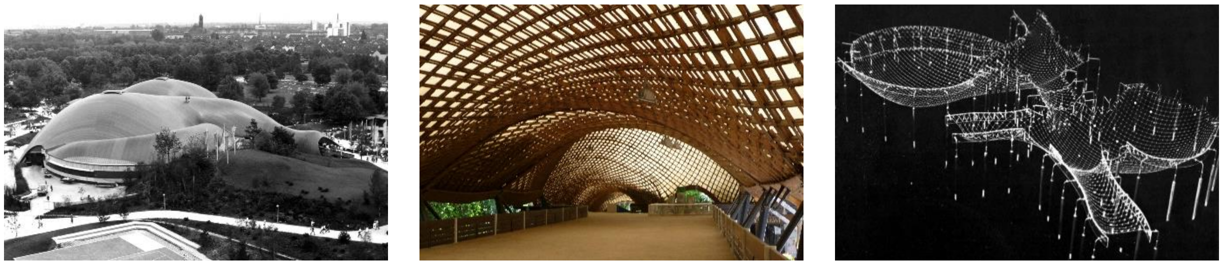
This section presents some of the results achieved using the framework presented earlier. It can be used to reproduce the design briefs for some famous funicular shapes, as well as more artificial examples. The case studies presented are summarized in Table 6.1, with the runtime of the optimizations needed for one solution of each structure.

The structures include approximations of the Mannheim Multihalle (referenced as “Mannheim” later on), a dome shape formed of two arches meeting in the dome center at a 90° angle (“oculus”), the Teshima Art Museum (“Teshima”), the California Academy of Sciences (“California”), and the Montevideo port warehouse (“Gaussian vaults”).

The Mannheim Multihalle, built in Mannheim Germany in 1974 and designed by Frei Otto, Carlfried Mutschler, and Winfried Langner, is an exhibition space initially built temporarily to house the 1975 Bundesgartenshau and kept later on (Happold & Lidell, 1975). It is an elastic gridshell made of hemlock wood, spanning around 60 by 60 meters. The shape was selected to be close to funicular – initial physical form-finding experiments were done with a network of hanging chains, as it was imagined the wood would creep too much under the permanent bending moments of a non-funicular shape. See Figure 6.3.

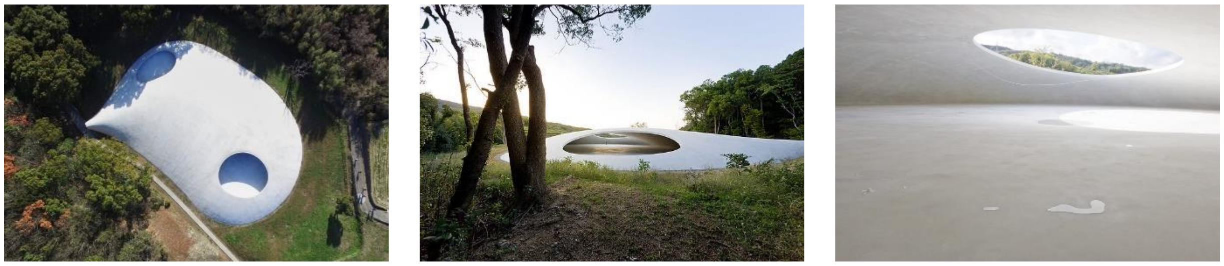
Table 6.1: Timings to generate one optimization result in each of the presented examples. Measured on a 2.8 GHz i7-7700HQ CPU.

	Nodes count	**Edges count**	**Runtime [s]**
Mannheim	705	1080	9.7
Oculus	4681	9060	5.9
Teshima	3313	1260	7.3
California	1684	3120	1.2
Gaussian vaults	880	1620	0.8



© Atelier Frei Otto Warmbronn   From Wikimedia, public domain.   (Burkhardt & Otto, 1978)

Figure 6.3: The Mannheim Multihalle, Germany, 1974, Frei Otto arch. From left to right, the complete structure, covered with its PVC envelope, the gridshell as seen on the inside, and form-finding models.



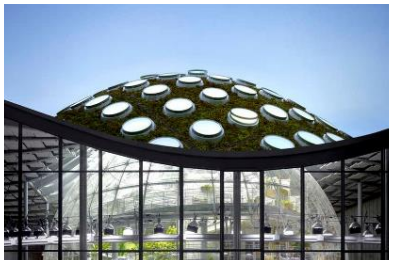
© Thomas Seear-Budd   © Iwan Baan   © Office of Ryue Nishizawa

Figure 6.4: The Teshima Art Museum, Kagawa, Japan, 2010. Nishizawa arch.

The quad-arched dome shape is an invention for this work, representing a simple dome where the designer wishes to have an opening on the top. It is modeled on a 20 meters wide base, with the arches each forming a 10 meters boundary on the ground. See Section 6.4.2 images.

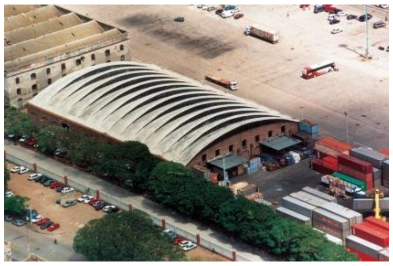
The Teshima Art Museum, built in 2010 on the island of Teshima, Kagawa Prefecture, Japan, and designed by Ryue Nishizawa, hosts a single piece of artwork – Matrix, created by sculptor Rei Nato. It is a thin concrete shell spanning about 40 by 60 meters, with a maximum height of 4.5 meters, see Figure 6.4. The shape was optimized (using the control points of a NURBS surface as the variables) to reduce strain energy while reproducing the architect's target shape (Sasaki, 2014); however, it is not clear how much more efficient the structure could be made (by being funicular), nor by how much it differs from the original design.

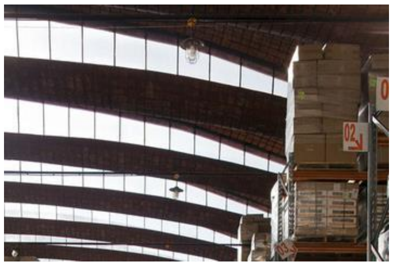
The California Academy of Sciences, for which a new building was erected in 2008 in San Francisco, California, is a research institute and one of the world's largest natural history museum (Wels, 2008). The building was designed by architect Renzo Piano, in collaboration with SWA Group. It is about 160 meters long by 90 meters wide and is covered by a hill-landscaped green roof of 2.5 acres. The tallest "hills" on this roof are about 20 by 20 meters domes with a squared base, and perforated with circular skylights to a varying degree. See Figure 6.5.


© Tim Griffith

© WolfmanSF @ Wikimedia

Figure 6.5: The California Academy of Sciences, San Francisco, CA, USA. Renzo Piano & SWA Group arch.


© Cámara Uruguya de Logística


© Leonardo Finotti

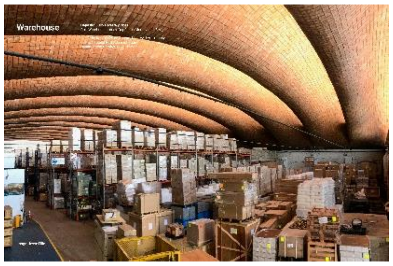

© Jesse Eliot

Figure 6.6: The Montevideo port warehouse, Uruguay, 1979. Dieste arch

The Montevideo port warehouse is one of several similarly shaped roofs – so-called non-continuous gaussian roofs – designed by Eladio Dieste from 1970 (Anderson et al., 2004). The warehouse was built in 1979, is made of 14 double-curvature shells, each 7 meters wide and spanning close to 50 meters, with a maximum rise of 8 meters. Each shell has an S-shaped cross-section, so that successive shells will slightly overlap with a vertical gap. Through that gap, light can pass through and illuminate the space below. See Figure 6.6.

6.4.1 Mannheim Multihalle

First, we reproduced the design brief for the elastic gridshell of the Mannheim Multihalle: a shape that is funicular under its own weight, covered by a grid of regular spacing. We reproduced the dimensions of the structure faithfully, loaded it by a constant weight, and optimized for a target member length of 4 m initially, which is reproducing 1/8th of the original grid density. To find this optimum, the objective function we used in $(P_{6.1})$ is simple:

$$f(\mathbf{x}) = \sum_{\boldsymbol{i}} \left(\boldsymbol{l_i} - \boldsymbol{l_i^{target}}\right)^{\mathbf{2}}. \tag{6.4}$$

The results of the simulations are shown in Figure 6.7, and the code this corresponds to in our system is shown in Figure 6.8. While a simple force density method simulation with uniform force densities shows a smooth shape, we can see that the member lengths this leads to a very spread out around 4 m; this would prevent the structure to be built with an elastic gridshell from a flat regular grid, as was done in reality. Hence to find a shape that could be built in this way, we initially optimize for all the members to be the same 4 m length. While we can achieve a very tight grouping of the lengths around this value, this damages the shape significantly, with many regions becoming too sharp. A possible solution is to allow the lengths to take one of two values – as was done in reality with some irregular rows in the grid, and optimize for minimum variations around these lengths. This is what we did in the bottom row of Figure 6.7, and we effectively get back to a smoother shape while still seeing minimal length variations.

Moving from the first optimization to the second is only a small modification in the objective function code, and results will be seen only a few seconds later as the simulation is run again. These small changes of optimization functions demonstrate the flexibility of our system, something that is not usually possible in typical inverse form-finding systems.

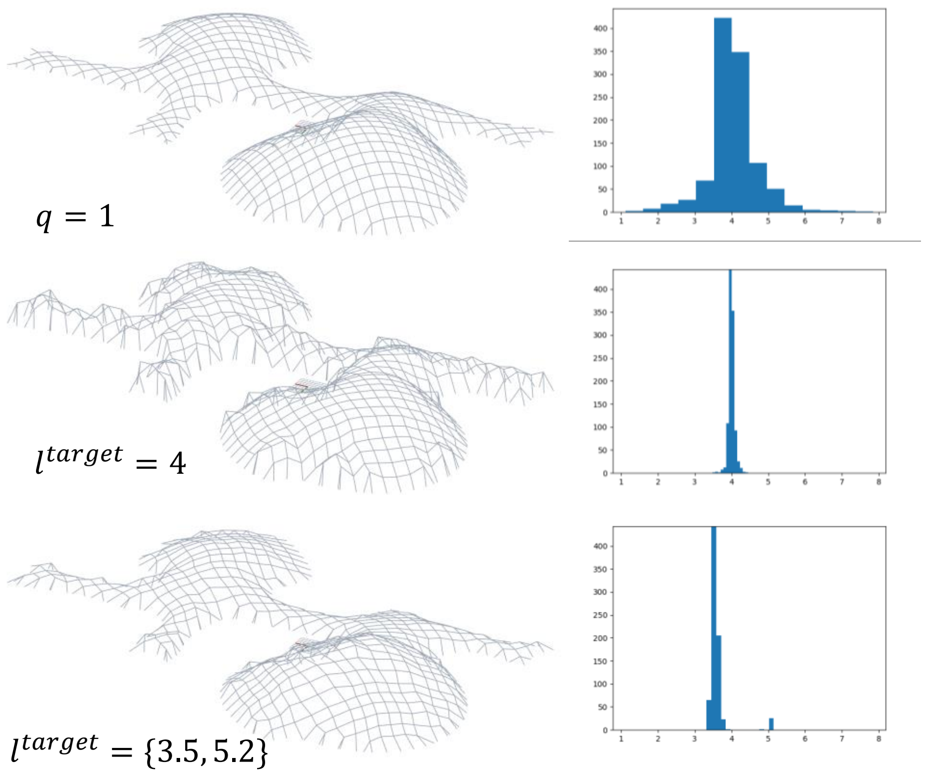


Figure 6.7: Equilibrium positions (left) and member length distribution (right) for a funicular surface reproducing the Mannheim Multihalle shape. The first row is a simple force density method result with uniform force densities, the second optimizes for a single 4 m grid size, the third allows members of length 3.5 m or 5.2 m.

```
def fix_lengths(connectivity_fixed, connectivity_free, ordered_nodes, **kwargs):
    n_nodes_fixed = connectivity_fixed.shape[1]
    diffs = sp.dot(connectivity_fixed, ordered_nodes[:n_nodes_fixed, :]) + sp.dot(
        connectivity_free, ordered_nodes[n_nodes_fixed:, :]
    )
    lengths_sq = np.sum(diffs ** 2, axis=1)
    return np.sum((lengths_sq - target_sq) ** 2) / target_sq ** 2
```

Figure 6.8: Objective function code for a uniform member length optimization.

6.4.2 Quad-arch with an oculus

Next, we imagined variations around finding a four-arched funicular structure with an opening in its center. We achieve this by penalizing points that have their equilibrium position in the center of the arch, using the following objective function (see code in Figure 6.9):

$$f(\mathbf{x}) = \sum_i \max\left(e^{\frac{d^2(x_i,0)}{R^2}},\, e^{-1}\right). \tag{6.5}$$

The results are presented in Figure 6.10. Again, simple modifications of this objective function, either through changing a parameter value or by combining it with additional objectives, lead to a variety of shapes. Because allowing for this flexibility does not slow down the optimization, or require the designer to carry complicated gradient calculations, this allows for a novel way of exploring the design space of funicular structures.

```
cylinder = lambda x: np.sum((x - center) ** 2, axis=1) / radius ** 2
bump = (
    lambda x: np.sum(np.clip(np.exp(-cylinder(x) ** 2 / two_sigma_sq), clip, None))
    - clip
)

def avoid_cylinder(
    connectivity_fixed, connectivity_free, ordered_nodes, stiffnesses, **kwargs
):
    return np.sum(bump(ordered_nodes[:, :2]))
```

Figure 6.9: Objective function code for creating an opening in a structure.

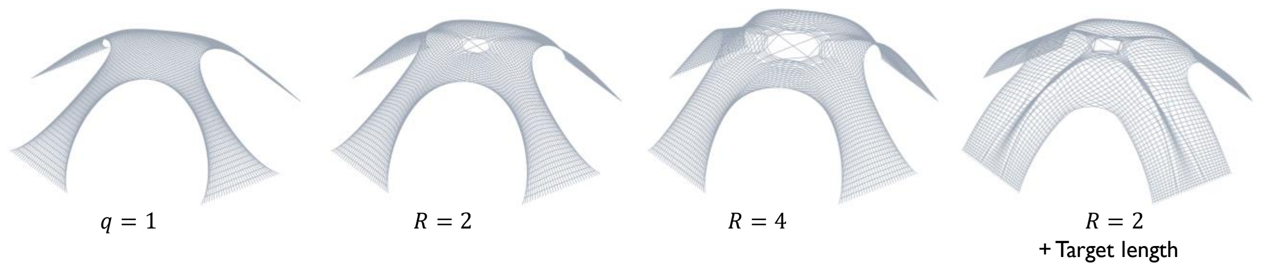


Figure 6.10: Equilibrium position results for the four-arched dome. From left to right, results of a simple force density method with uniform force densities, opening with a target size of 2 m, a target size of 4 m, and 2 m opening with an additional uniform member length objective.

6.4.3 Teshima Art Museum

For the Teshima Art Museum's structure, a more complex objective function was needed. This time, two openings (of the same form as in the previous section) are added, in addition to a score giving the vertical distance from the equilibrium position to a shape representing the general cross-section of the structure as it was built, and a linear slope from the entrance to the larger opening. Figure 6.11 shows an image of the result that can be obtained, showing a good reproduction of the shell as it was built.

In Figure 6.12 are shown some iterations of the optimization process. The openings form early, then the optimizer improves mostly on the second part of the objective function, giving its shape to the structure. While the openings still have some nodes and mesh edges in them, the force densities values show that they are barely loaded, and in fact only remain there as an artifact of the bounds that were given to the optimizer on the force densities.

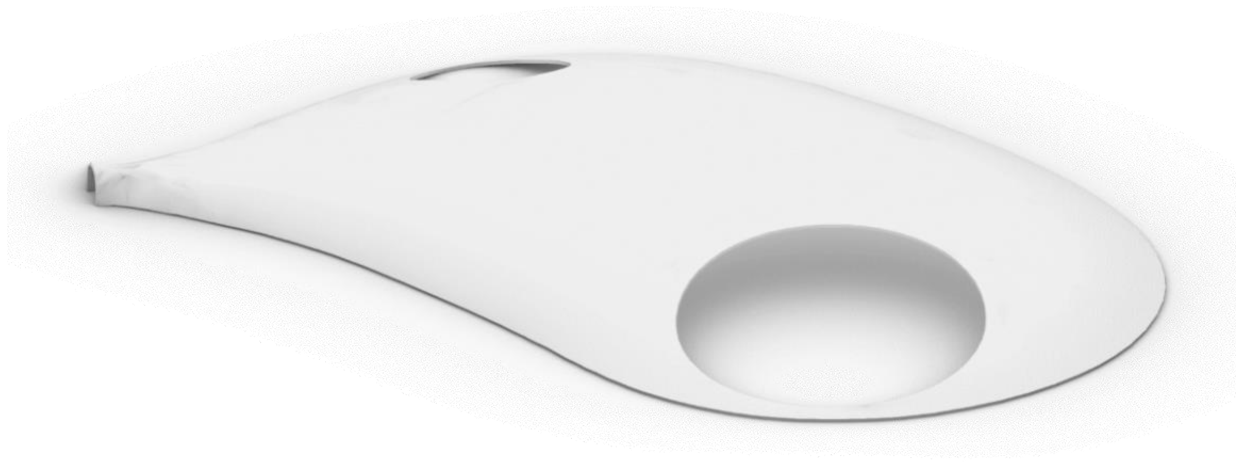


Figure 6.11: Rendering of the funicular shape obtained with this work to reproduce the Teshima museum shape.

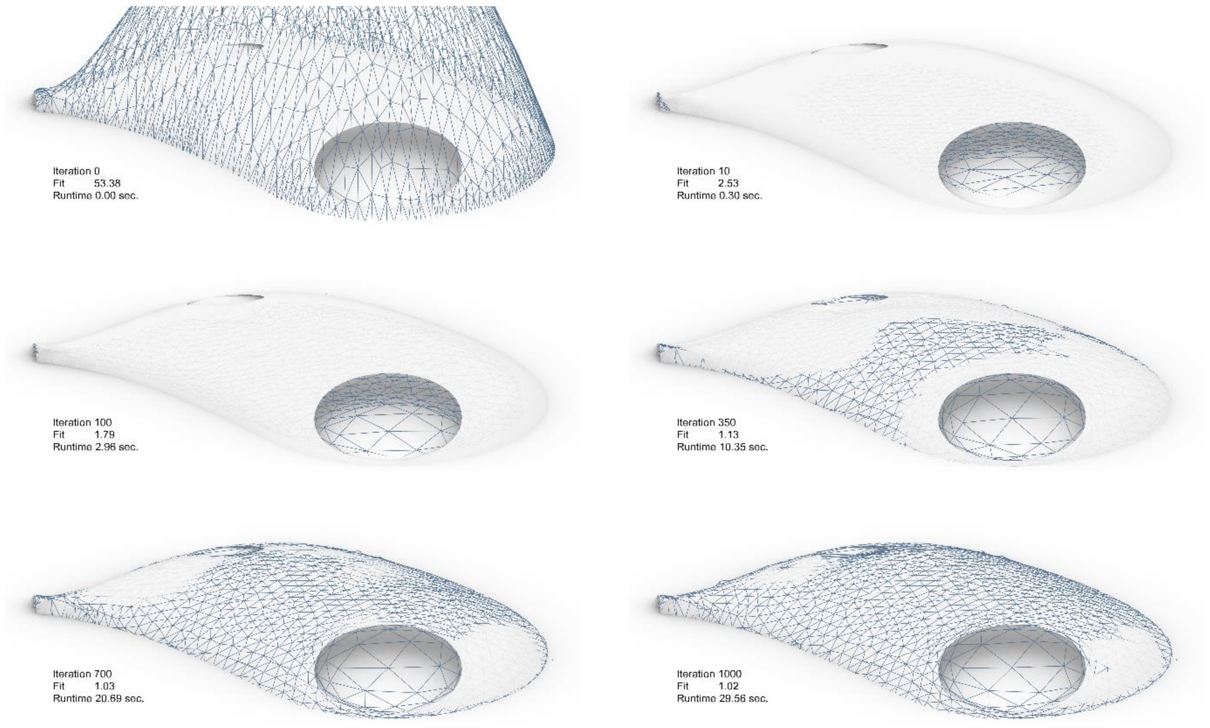


Figure 6.12: Some iterations during the optimization to obtain the Teshima shape. The fit value is the value objective function relative to the final value found at the optimum.

6.4.4 California Academy of Sciences

Extending on the idea of openings in a funicular roof, we reproduce the large "hills" on the green roof of the California Academy of Sciences. Each of them is roughly a 20-by-20-meter square dome, with many openings for skylights. By introducing many objectives for openings like in Section 6.4.2, and moving them around the surface, a variety of shapes can be obtained. Figure 6.13 shows one such example, and Figure 6.14 some variations on the positions. Along with this study, it is possible to get an estimation of the relative structural efficiency of the meshes by comparing the elastic energy scores:

$$\sum_i |F_i l_i| = \sum_i q_i l_i^2 \,. \tag{6.6}$$

These scores could be included in the optimization, to guide it towards more efficient shapes.

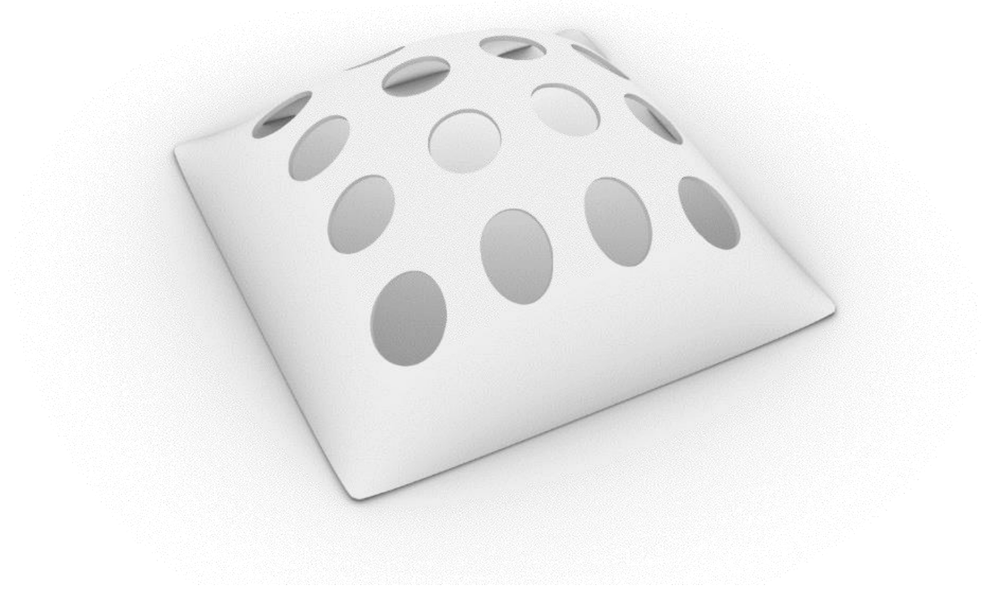


Figure 6.13: Rendering of a funicular square dome with multiple openings, reproducing the hills of the California Academy of Sciences roof.

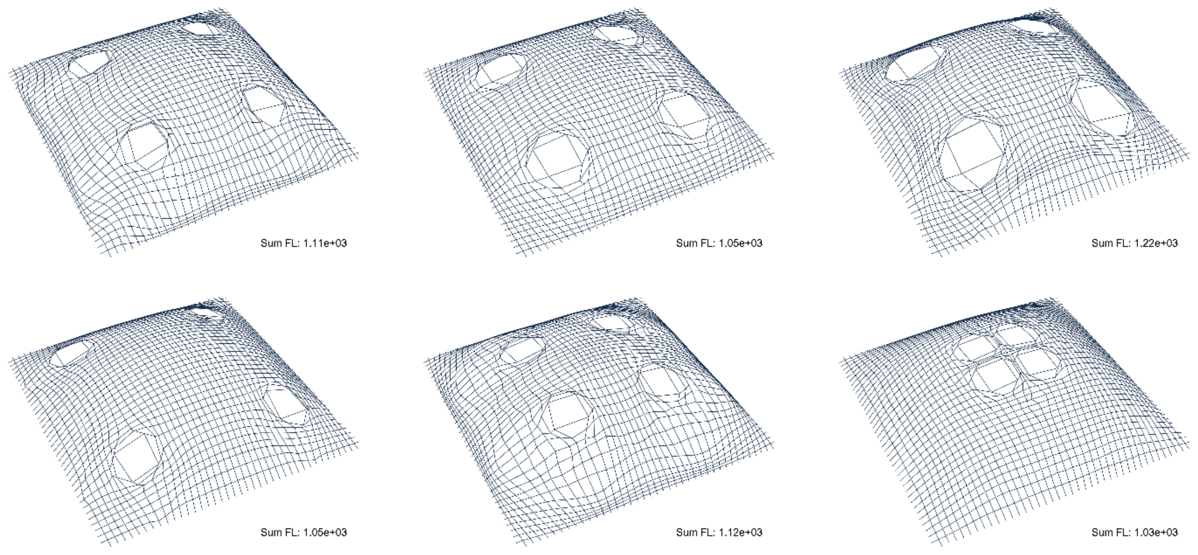


Figure 6.14: Some variations on the openings of the square dome. The scores on the bottom left represent a quick estimate of the relative structural efficiency of each dome, and are calculated as the sum of the forces times lengths in each member.

6.4.5 Gaussian vaults

Finally, we experiment with shell-beams shapes, like Dieste for his famous brick roofs of non-continuous gaussian shapes. The objective here was to give the designer funicular shapes that could be varied easily in the shape of the roof they realize, their maximum height, and the size of the vertical gap between two successive beams.

We achieve the roof shaping by simply changing the shape of the boundary that one beam attaches to, which is simply two lines of supports opposite each other, spanning the roof in its short dimension. The maximum height of the shell and the vertical gap size are the result of the combination of three objectives. The first one minimizes the error between the obtained vertical gap and its target value given by the designer, the second the elastic energy, and the third the material use or total bar length of the mesh:

$$f = \left(\bar{h} - \bar{h}_0\right)^2 + \alpha\left(\sum_i q_i l_i^2\right)^n + (1-\alpha)\left(\sum_i l_i^2\right)^m, \qquad (6.7)$$

Where α are weighting parameters between the last two objectives, $\bar{h}$ the obtained vertical distance between the two extreme nodes at the mid-span of the beam, $\bar{h}_0$ the target for $\bar{h}$, and n, m arbitrary scaling parameters.

Generally, the first term makes the gap close to its input target, the second term gives the beam a high rise and bulbous dome shape, and the third term makes it taught and flat. By changing the weighing parameter α, the shape will change between a shallow shell and a deep one. By changing the target height difference $\bar{h}_0$, the front and back nodes at the mid-span of the beam will rise and fall accordingly. Because of the additional terms effectively acting as regularization terms, the rest of the mesh will follow and give a smooth shape, rather than a pointy one with only two nodes matching the objective. This is the reason for the additional n and m scaling parameters, which were varied from 1 to 4 to produce better regularization terms.

Figure 6.15 gives some examples of possible results.

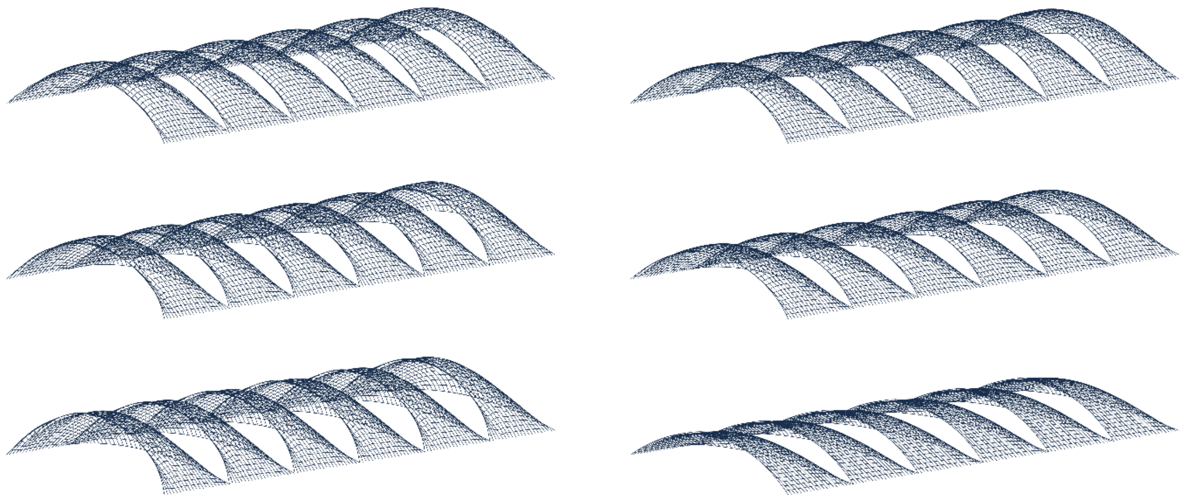


Figure 6.15: Variations on the Dieste beam shapes. Left column varies the height difference target, right column changes the weighting between structural efficiency and material use.

6.5 Conclusion

In conclusion, this chapter demonstrated a framework for the easy combination of multiple objectives into a funicular shapes generation scheme. The funicular shapes are given by a force density method solver, and the objectives are optimized using a gradient-based general-purpose optimizer. Because the framework is implemented in an Automatic Differentiation package, there is very little cost to adding new objectives or recombining them differently, yet the optimization process remains fast and stable. This lets the designer easily explore a vast design space of funicular shapes, guiding their exploration by the objectives they choose.

In future work, it would be interesting to extend our design tool and make its use easier by implementing a catalog of possible objectives that the designer could pick from, instead of coding. Simple combinators like in Section 6.4 can be added to make building a combined objective easy. Additionally, it would be interesting to use multi-objective optimization techniques to explore a larger expanse of the design space. Similarly, human-guided optimization could let the designer change the exploration direction during an optimization.

This also demonstrates how efficient automatic differentiation and general-purpose optimizers can be when combined, and applying these techniques to other form-finding problems could lead to the creation of powerful design tools.

7 Conclusion

7.1 Summary of contributions

This dissertation has focused on improving form-finding workflows by giving more control over the obtained shapes to the designer. The main contributions were applied to bending-active structures and funicular structures design, and implemented inverse form-finding workflows using a nested-optimizers strategy. The inner optimizer is a specially selected direct form-finding solver, the outer optimizer is a general-purpose optimization routine that attempts to match the results of the forward simulation to the design intent.

In general, the novelty in this dissertation lies in the focus on inverse form-finding workflows, and how they can be made to perform well enough for use in interactive design. By defining a clear general framework on the implementation of such workflows in a nested optimizer loop, the key requirements on each were articulated. This was demonstrated with case studies of two mechanical systems.

For bending-active structures, the performance (speed, accuracy, reliability) of direct form-finding solvers was measured, and recommendations on their use in an inverse form-finding setup given. Because the outer optimization loop in this setup needs to rely on a robust forward simulation with minimal configuration during the optimization, general-purpose optimizers like SLSQP and L-BFGS perform better than usual choices in bending-active simulation like dynamic relaxation. Using this insight, an inverse form-finding workflow was built and applied to simple elastica-like arc lamp structures, and the closest-fit optimization of an elastic gridshell. Experimenting with different formulations of the problem, especially through the definition of its variables, their influence on the final results was demonstrated.

In funicular structures, because robust and efficient direct form-finding solvers are readily available, the focus was directly taken on an inverse workflow. Initially focusing on a closest-fit to target surface situation, closed-form formulations of gradients and hessian were derived, which lead to a fast and stable optimization. This also demonstrated how finding closed-form expressions of these derivatives is a major blocking point in creating more versatile inverse form-finding workflows.

This same inner optimizer was reimplemented in an Automatic Differentiation framework, to produce an inverse form-finding tool for funicular surfaces that was not limited by this issue. Because this is a novel way of implementing such workflows, this exposed how the design intent can be represented by more complex objects than a target surface. Reproducing several case studies of existing funicular structures, it is shown how this lets the designer explore new portions of their design space, and how a fine control over the objectives of the inverse form-finding optimization leads to possible incremental modifications of the design.

For the tools that are necessary to implement these workflows, a key requirement that was found during the work presented in this dissertation, is that they should be compatible with tools that are well-known by designers. This lets designers use a familiar interface for the initial modeling tasks and the display and post-processing of the results. To achieve this, a communication layer between Rhino/Grasshopper and Python was created, allowing the use of advanced scientific programming tools from Python (scientific computing with Numpy, just-in-time compilation with Numba, automatic differentiation with Autograd), in the well-known Rhino environment. This was achieved with GH Python Remote (Cuvilliers & Mueller, 2017), is aimed at being as reusable as possible, and so far has been downloaded around 25,000 times (https://pypi.org/project/gh-python-remote/).

7.2 Potential impact

In building inverse form-finding tools, we hope to give the designer more control on the shapes of complex structural systems, that are governed by specific rules limiting their design spaces. Because these shapes adapt to bespoke construction systems, they can be chosen to very efficiently use material. Funicular shapes for example, will very efficiently span large spaces. Bending-active structures are very adaptable and have a large potential for reuse. Additionally, thanks to this added control, designers can explore larger regions of the design space of their chosen construction system, leading to more variety in structures without any penalty on efficiency. While the shapes of many shells were initially chosen as pre-defined mathematical functions like revolution surfaces of lines or simple curves, so that their analysis would be simpler, we now have tools to create and analyze free-form funicular shells.

This new control on the form-finding process will also give designers more freedom in how they use these tools. While existing direct form-finding tools tend to give results of a similar architectural style, designers can now guide the exploration towards target shapes and author their own style on them.

7.3 Future work

7.3.1 Current limitations and next steps

For each of the contributions of this dissertation to inverse form-finding of bending-active structures and funicular structures, there are immediate next steps that could expand the quality of the tools.

In bending-active structures, the solver presented could benefit a lot from the same Automatic Differentiation treatment as was done for funicular structures. The key challenge here is that the inner optimizer used in bending-active structures is more involved, and if applied carelessly the Automatic Differentiation framework has the potential for greatly slowing down the computation. This should still be possible as it is within the realm of what Automatic Differentiation can tackle, and similar systems have been implemented in Automatic Differentiation frameworks (Degrave et al., 2017). Being able to optimize for different metrics than the closest fit to a surface would be hugely beneficial for elastic gridshells.

In funicular structures, it would be interesting to expand the scope of the inverse form-finding objectives to produce a catalog that the designer can pick from. This could be compiled into a graphical user interface so that the objectives can be combined easily; an even more powerful interface would allow the designer to paint such goals on the regions of the structure where they should apply. Indeed, in the current setup this kind of geographical information needs to be manually coded in the objective function, which is not a very reusable design.

Finally for the presented inverse form-finding tools, it would make sense to compare these implementations to existing ones, by reimplementing the specific inverse form-finding that they solve in frameworks presented in this dissertation. Although the latter are less tailored towards one specific problem than the former, they need to be competitive so that they are useful on simpler separate problems just as much as on more complex ones. Both of the implementations presented still suffer from challenging computational power requirements, and any improvement there will be beneficial. They also tend to require minute manual tuning of the parameters of the outer optimizer, something that is not desirable in general for a design tool offered for a wider audience. Additional experiments will be required to find robust optimizers for this.

7.3.2 Open questions

More open questions remain for future work, especially in the area of expending inverse form-finding solvers to generalist frameworks both in terms of the mechanics-based constraints they can encode, and design intent objectives. This is similar to the shift that happened for direct form-finding solvers with the development of tools like Kangaroo.

For inverse form-finding, two important pieces of software engineering would probably be required to make such a setup work. First, the direct form-finding solver would need to be implemented in an Automatic Differentiation framework, as we have described previously, so that general design objectives could be efficiently solved for. Additionally, we expect that the mechanical equilibrium constraints will need to be compiled in such a way that computing one value of the energy function does not need to loop through the objects of an object-oriented programming system. In fact, this is already a major roadblock in the implementation of generalist direct form-finding software, and would be potentially slowed down even further if implemented in an Automatic Differentiation framework.

Additionally, it will be interesting to run user studies on the resulting design tools, showing how inverse form-finding tools can help create a faster, more directed design process, and in general give back control to the designer on form-finding operations. This should also demonstrate how the speed of the design process and the ease of options comparison help keep the design process flowing, with more engagement from the designer.

Finally, these systems can be made easier to use by attaching to them a cookbook of usual systems modeled. This lets designers pick and match options as they apply to their particular case. A lot of the time, building the program that represents the building system constraints is difficult, but repeatable, so this would save time. It would also help in avoiding incorrect formulations, and reduce the complexity of the tuning of the outer optimizer that was mentioned in the previous section.

7.4 Concluding remarks

In conclusion, this dissertation showed how the ubiquitous problem of inverse form-finding in architecture is made apparent, yet unsolved, by the recent proliferation of direct form-finding tools. All design systems strive to let the designer stay as true as possible to their original intent, and convey this meaning through an instance of a construction system realizing the intent and giving it a shape. Because it removes from the hands of the designer almost all control on the resulting shape, direct form-finding alone can be a limited design tool. In expert hands, direct form-finding has lead to the design of many great structures; when more control is gained over its results through inverse form-finding method it can be used by a wider audience and in more constrained contexts.

By producing designers with well-crafted inverse form-finding solutions, they are empowered again to design with complex mechanical equilibrium constraints, and can explore a wider range of their design spaces. This dissertation realized a comprehensive overview of inverse form-finding systems pointing to a way forward in the direction of better design tools for contemporary construction systems.

8 References

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D. G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., … Zheng, X. (2016). TensorFlow: A system for large-scale machine learning. *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI '16)*, 265–283.

Adriaenssens, S., & Barnes, M. R. (2001). Tensegrity spline beam and grid shell structures. *Engineering Structures*, *23*(1), 29–36. https://doi.org/10.1016/S0141-0296(00)00019-5

Adriaenssens, S., Block, P., Veenendaal, D., & Williams, C. (2014). *Shell Structures for Architecture: Form Finding and Optimization*. Routledge.

Ahlquist, S., & Menges, A. (2013). Frameworks for Computational Design of Textile Micro-Architectures and Material Behavior in Forming Complex Force-Active Structures. *ACADIA 13: Adaptive Architecture*, 281–292.

Anders, C., Deleuran, H., Quinn, G., Piker, D., Pearson, W., Brandt-Olsen, C., Naicu, D., & Lewis, H. (2016). Calibrated Modelling of Form-Active Hybrid Structures. *Workshops of the Smart Geometry Conference*. Smart Geometry, Gothenburg, Sweden.

Anderson, S., Dieste, E., & Hochuli, S. (2004). *Eladio Dieste: Innovation in Structural Art*. Princeton Architectural Press.

Audoly, B., & Pomeau, Y. (2010). *Elasticity and Geometry: From hair curls to the nonlinear response of shells*. Oxford University Press. https://doi.org/10.1142/9789812792778_0001

B. Speelpenning. (1980). *Compiling Fast Partial Derivatives of Functions Given by Algorithms* [Ph.D. Dissertation]. University of Illinois at Urbana-Champaign.

Bagneris, M., Motro, R., Maurin, B., & Pauli, N. (2008). Structural Morphology Issues in Conceptual Design of Double Curved Systems: *International Journal of Space Structures*, *23*(2), 79–87. https://doi.org/10.1260/026635108785260560

Bagrianski, S., & Halpern, A. B. (2014). Form-finding of compressive structures using Prescriptive Dynamic Relaxation. *Computers & Structures*, *132*, 65–74. https://doi.org/10.1016/j.compstruc.2013.10.018

Ban, S. (2003). The Japanese pavilion. In M. McQuaid (Ed.), *Shigeru Ban* (pp. 8–11). Phaedon.

Barnes, M. R. (1988). Form-finding and analysis of prestressed nets and membranes. *Computers & Structures*, *30*(3), 685–695. https://doi.org/10.1016/0045-7949(88)90304-5

Barnes, M. R. (1999). Form Finding and Analysis of Tension Structures by Dynamic Relaxation. *International Journal of Space Structures*, *14*(2), 89–104. https://doi.org/10.1260/0266351991494722

Barnes, M. R. (1975). Applications of dynamic relaxation to the topological design and analysis of cable, membrane and pneumatic structures. *2nd International Conference on Space Structures*, 211–219.

Barnes, M. R., Adriaenssens, S., & Krupka, M. (2013). A novel torsion/bending element for dynamic relaxation modeling. *Computers & Structures*, *119*, 60–67. https://doi.org/10.1016/j.compstruc.2012.12.027

Barthelemy, J.-F. M., & Hall, L. E. (1995). Automatic differentiation as a tool in engineering design. *Structural Optimization*, *9*(2), 76–82. https://doi.org/10.1007/BF01758823

Baverel, O., Caron, J.-F., Tayeb, F., & du Peloux, L. (2012). Gridshells in composite materials: Construction of a 300 m2 forum for the Solidays' festival in Paris. *Structural Engineering International*, *22*(3), 408–414. https://doi.org/10.2749/101686612X13363869853572

Baydin, A. G., Pearlmutter, B. A., Radul, A. A., & Siskind, J. M. (2018). Automatic Differentiation in Machine Learning: A Survey. *Journal of Machine Learning Research*, *18*, 1–43.

Beda, L. M., Korolev, L. N., Sukkikh, N. V., & Frolova, T. S. (1959). *Programs for automatic differentiation for the machine BESM* [Technical Report]. Institute for Precise Mechanics and Computation Techniques, Academy of Science.

Bergou, M., Audoly, B., Vouga, E., Wardetzky, M., & Grinspun, E. (2010). Discrete viscous threads. *ACM Transactions on Graphics*, *29*(4), 1. https://doi.org/10.1145/1833351.1778853

Bergou, M., Wardetzky, M., Robinson, S., Audoly, B., & Grinspun, E. (2008). Discrete elastic rods. *ACM Transactions on Graphics*, *27*(3). https://doi.org/10.1145/1360612.1360662

Bergstra, J., Breuleux, O., Bastien, F., Lamblin, P., Pascanu, R., Desjardins, G., Turian, J., & Bengio, Y. (2010, June). Theano: A CPU and GPU Math Expression Compiler. *Proceedings of the 9th Python in Science Conference*. SciPy, Austin, TX.

Birgin, E. G., & Martínez, J. M. (2008). Improving Ultimate Convergence of an Augmented Lagrangian Method. *Optimization Methods Software*, *23*(2), 177–195. https://doi.org/10.1080/10556780701577730

Block, P. (2009). *Thrust Network Analysis: Exploring three-dimensional equilibrium* [Ph.D. Dissertation, Massachusetts Institute of Technology]. https://dspace.mit.edu/handle/1721.1/49539

Block, P., & Lachauer, L. (2011). *Closest-Fit, Compression-Only Solutions for Freeform Shells*. 35th Annual Symposium of IABSE / 52nd Annual Symposium of IASS / 6th International Conference on Space Structures, London, UK.

Block, P., & Ochsendorf, J. (2007). Thrust network analysis: A new methodology for three-dimensional equilibrium. *Journal of the International Association for Shell and Spatial Structures*, *48*(3), 8.

Block, P., Van Mele, T., Méndez Echenagucia, T., Hofmann, H., Ochsendorf, J., DeJong, M., & Giardina, G. (2016). *Droneport Prototype—Venice Architecture Biennale 2016*. Block Research Group. https://block.arch.ethz.ch/brg/project/venice-biennale-2016_droneport

Bouaziz, S., Deuss, M., Schwartzburg, Y., Weise, T., & Pauly, M. (2012). Shape-up: Shaping discrete geometry with projections. *Computer Graphics Forum*, *31*(5), 1657–1667. https://doi.org/10.1111/j.1467-8659.2012.03171.x

Bouaziz, S., Martin, S., Liu, T., Kavan, L., & Pauly, M. (2014). Projective Dynamics: Fusing Constraint Projections for Fast Simulation. *ACM Transactions on Graphics*, *33*(4), 154–165. https://doi.org/10.1145/2601097.2601116

Bouhaya, L., Baverel, O., & Caron, J.-F. (2014). Optimization of gridshell bar orientation using a simplified genetic approach. *Structural and Multidisciplinary Optimization*, *50*(5), 839–848. https://doi.org/10.1007/s00158-014-1088-9

Brew, J. S., & Brotton, D. M. (1971). Non-linear structural analysis by dynamic relaxation. *International Journal for Numerical Methods in Engineering*, *3*(4), 463–483. https://doi.org/10.1002/nme.1620030403

Brew, J. S., & Lewis, W. J. (2007). Free hanging membrane model for shell structures. *International Journal for Numerical Methods in Engineering*, *71*(13), 1513–1533. https://doi.org/10.1002/nme.1976

Brown, N. C. (Nathan C. (2019). *Early building design using multi-objective data approaches* [Ph.D. Dissertation, Massachusetts Institute of Technology]. https://dspace.mit.edu/handle/1721.1/123573

Bubner, E. (1972). *Zum Problem der Formfindung vorgespannter Seilnetzflächen* [Ph.D. Dissertation]. University of Stuttgart.

Burkhardt, B., & Otto, F. (1978). *Multihalle Mannheim (IL 13)*. Freunde und Förderer der Leichtbauforschung.

Byrd, R. H., Lu, P., Nocedal, J., & Zhu, C. (1995). A Limited Memory Algorithm for Bound Constrained Optimization. *SIAM J. Sci. Comput.*, *16*(5), 1190–1208. https://doi.org/10.1137/0916069

Chilton, J., Macdonald, A., & Pedreschi, R. (2000). *The Engineer's Contribution to Contemporary Architecture: Heinz Isler*. Thomas Telford Publishing. https://doi.org/10.1680/eccahi.28784

Coar, L. (2010). *On the road / En route*. http://ontheroadenroute.blogspot.ca

Coar, L. (2012). Wobbly Structures: Exploring the potentials of flexible frames & fabric formed ice structures. *International Conference on Flexible Formwork*, 1–13.

Coar, L., Cox, M., Adriaenssens, S., & De Laet, L. (2017). The design and construction of bending active framed fabric formed ice shells utilizing principle stress patterns. In A. Bögle & M. Grohmann (Eds.), *Interfaces: Architecture. Engineering. Science. Proceedings of the IASS annual symposium 2017*.

Crane, K. (2014). *Discrete differential geometry: An applied introduction* [Lecture notes]. https://www.cs.cmu.edu/~kmcrane/Projects/DDG/paper.pdf

Cuvilliers, P., Douthe, C., du Peloux, L., & Le Roy, R. (2017). Hybrid structural skin: Prototype of a GFRP elastic gridshell braced by a fiber-reinforced concrete envelope. *Journal of the International Association for Shell and Spatial Structures*, *58*(1), 65–78. https://doi.org/10.20898/j.iass.2017.191.853

Cuvilliers, P., Mayencourt, P., & Mueller, C. (2018, September). Design and fabrication of bending-active structures with controlled shapes: The arc lamp. *Workshops of the Advances in Architectural Geometry Conference 2018*.

Cuvilliers, P., & Mueller, C. (2017). *GH Python Remote*. Digital Structures Github. https://github.com/Digital-Structures/ghpythonremote

D'Amico, B., Kermani, A., Zhang, H., Shepherd, P., & Williams, C. J. K. (2015). Optimization of cross-section of actively bent grid shells with strength and geometric compatibility constraints. *Computers & Structures*, *154*, 163–176. https://doi.org/10.1016/j.compstruc.2015.04.006

de Goes, F., Alliez, P., Owhadi, H., & Desbrun, M. (2013). On the equilibrium of simplicial masonry structures. *ACM Transactions on Graphics*, *32*(4), 1. https://doi.org/10.1145/2461912.2461932

Degrave, J., Hermans, M., Dambre, J., & Wyffels, F. (2017). A Differentiable Physics Engine for Deep Learning in Robotics. *Workshops of the International Conference on Learning Representations*. International Conference on Learning Representations, Toulon, France. https://openreview.net/forum?id=HkrB8XXte

del Blanco García, F. L., & García Ríos, I. (2019). Algorithm Design for Ruled Surfaces. Case Study of Felix Candela. In C. L. Marcos (Ed.), *Graphic Imprints* (pp. 1577–1585). Springer International Publishing. https://doi.org/10.1007/978-3-319-93749-6_131

Deprez, L. (1968). *Shape optimization for arch dams* [M.Sc. Thesis, Massachusetts Institute of Technology]. https://dspace.mit.edu/handle/1721.1/87810

Dessi-Olive, J. (2017). *Computing with matter, shapes, and forces: Toward material and structural primacy in architecture* [M.Sc. Thesis, Massachusetts Institute of Technology]. https://dspace.mit.edu/handle/1721.1/111705

Douthe, C. (2007). *Étude de structures élancées précontraintes en matériaux composites: Application à la conception des gridshells* [Thèse de doctorat, École Nationale des Ponts et Chaussées]. https://pastel.archives-ouvertes.fr/pastel-00003723

Douthe, C., Caron, J.-F., & Baverel, O. (2010). Gridshell structures in glass fibre reinforced polymers. *Construction and Building Materials*, *24*(9), 1580–1589. https://doi.org/10.1016/j.conbuildmat.2010.02.037

du Peloux, L., Tayeb, F., & Lefevre, B. (2015). Formulation of a 4-DoF torsion/bending element for the formfinding of elastic gridshells. *Future Visions: Proceedings of the International Association for Shell and Spatial Structures (IASS) Symposium 2015*.

Dudte, L. H., Vouga, E., Tachi, T., & Mahadevan, L. (2016). Programming curvature using origami tessellations. *Nature Materials*, *15*(5), 583–588. https://doi.org/10.1038/nmat4540

Eigensatz, M., Deuss, M., Schiftner, A., Kilian, M., Mitra, N., & Pauly, M. (2010). Case Studies in Cost-Optimized Paneling of Architectural Freeform Surfaces. *Advances in Architectural Geometry*, 49–72. https://doi.org/10.1007/978-3-7091-0309-8_4

Eigensatz, M., Sumner, R. W., & Pauly, M. (2008). Curvature-Domain Shape Processing. *Computer Graphics Forum*, *27*(2), 241–250. https://doi.org/10.1111/j.1467-8659.2008.01121.x

Engel, H. (1967). *Tragsysteme = Structure systems*. Deutsche Verlags-Anstalt.

Fleischmann, M., & Menges, A. (2012). ICD/ITKE Research Pavilion: A case study of multi-disciplinary collaborative computational design. In C. Gengnagel, A. Kilian, N. Palz, & F. Scheurer (Eds.), *Computational Design Modelling* (pp. 239–248). Springer. https://doi.org/10.1007/978-3-642-23435-4_27

Frankel, T. (2011). *The geometry of physics: An introduction* (3rd editio). Cambridge University Press.

Garg, A., Sageman-Furnas, A. O., Deng, B., Yue, Y., Grinspun, E., Pauly, M., & Wardetzky, M. (2014). Wire Mesh Design. *ACM Trans. Graph.*, *33*(4), 66:1–66:12. https://doi.org/10.1145/2601097.2601106

Germain, S., Scherer, M., & Steinmann, P. (2010). On Inverse Form Finding for Anisotropic Hyperelasticity in Logarithmic Strain Space. *International Journal of Structural Changes in Solids*, *2*(2), 1–16.

Goldenthal, R., Harmon, D., Fattal, R., Bercovier, M., & Grinspun, E. (2007). Efficient simulation of inextensible cloth. *ACM Transactions on Graphics*, *26*(3), 49. https://doi.org/10.1145/1276377.1276438

Govindjee, S., & Mihalic, P. a. (1998). Computational methods for inverse deformations in quasi-incompressible finite elasticity. *International Journal for Numerical Methods in Engineering*, *43*(5), 821–838. https://doi.org/10.1002/1097-0207

Goyal, S., Perkins, N. C., & Lee, C. L. (2008). Non-linear dynamic intertwining of rods with self-contact. *International Journal of Non-Linear Mechanics*, *43*(1), 65–73. https://doi.org/10.1016/j.ijnonlinmec.2007.10.004

Grafe, R., Gröbner, G., Gründig, L., Hennicke, J., Matsushita, K., Otto, F., Sataka, K., Schaur, E., Schock, H.-J., & Shirayanagi, T. (1974). Plane nets. In Institut für leichte Flächentragwerke (Ed.), *IL 10—Grid shells* (pp. 138–155). Karl Krämer Verlag.

Hablicsek, M., Akbarzadeh, M., & Guo, Y. (2019). Algebraic 3D graphic statics: Reciprocal constructions. *Computer-Aided Design*, *108*, 30–41. https://doi.org/10.1016/j.cad.2018.08.003

Hairer, E., Lubich, C., & Wanner, G. (2006). *Geometric numerical integration: Structure-preserving algorithms for ordinary differential equations* (Second edi). Springer Berlin Heidelberg New York.

Happold, E., & Lidell, W. I. (1975). Timber Lattice Roof for the Mannheim Bundesgartenschau. *The Structural Engineer*, *53*(3), 99–135.

Huerta, S. (2006). Structural Design in the Work of Gaudí. *Architectural Science Review*, *49*(4), 324–339. https://doi.org/10.3763/asre.2006.4943

Ingraham, J., Riesselman, A., Sander, C., & Marks, D. (2019). *Learning protein structure with a differentiable simulator*. International Conference on Learning Representations (ICLR).

Jacobson, A., Baran, I., Popović, J., & Sorkine, O. (2011). Bounded biharmonic weights for real-time deformation. *ACM Transactions on Graphics (TOG)*. https://dl.acm.org/doi/abs/10.1145/2010324.1964973

Jiang, C., Tang, C., Seidel, H.-P., & Wonka, P. (2017). Design and Volume Optimization of Space Structures. *ACM Trans. Graph.*, *36*(4), 159:1–159:14. https://doi.org/10.1145/3072959.3073619

Johnson, S. G. (n.d.). *The NLopt nonlinear-optimization package*. http://ab-initio.mit.edu/nlopt

Joshi, P. P. (2008). *Minimizing Curvature Variation for Aesthetic Surface Design* [Ph.D. Dissertation, EECS Department, University of California, Berkeley]. http://www2.eecs.berkeley.edu/Pubs/TechRpts/2008/EECS-2008-129.html

Kilian, A., & Ochsendorf, J. (2005). Particle-spring systems for structural form finding. *Journal of the International Association for Shell and Spatial Structures*, *46*(2), 77–84.

Kilian, M., Flöry, S., Chen, Z., Sheffer, A., & Pottmann, H. (2008). Developable Surfaces with Curved Creases. *Advances in Architectural Geometry*, 33–36.

Kmet, S., & Mojdis, M. (2015). Time-dependent analysis of cable nets using a modified nonlinear force-density method and creep theory. *Computers & Structures*, *148*, 45–62. https://doi.org/10.1016/j.compstruc.2014.11.004

Koohestani, K. (2014). Nonlinear force density method for the form-finding of minimal surface membrane structures. *Communications in Nonlinear Science and Numerical Simulation*, *19*(6), 2071–2087. https://doi.org/10.1016/j.cnsns.2013.10.023

Kovalsky, S. Z., Galun, M., & Lipman, Y. (2016). Accelerated quadratic proxy for geometric optimization. *ACM Transactions on Graphics*, *35*(4), 1–11. https://doi.org/10.1145/2897824.2925920

Kraft, D. (1988). A software package for sequential quadratic programming. *Forschungsbericht-Deutsche Forschungs- Und Versuchsanstalt Fur Luft- Und Raumfahrt*.

Levenberg, K. (1944). A Method for the Solution of Certain Problems in Least-Squares. *Quarterly Applied Mathematics 2*, 164–168.

Levien, R. (2008). *The elastica: A mathematical history* (Issue UCB/EECS-2008-103).

Lienhard, J. (2014). *Bending-Active Structures: Form-finding strategies using elastic deformation in static and kinetic systems and the structural potentials therein* [Dr.Ing. Thesis]. University of Stuttgart.

Lienhard, J., Alpermann, H., Gengnagel, C., & Knippers, J. (2013). Active Bending, A Review on Structures where Bending is used as a Self-Formation Process. *International Journal of Space Structures*, *28*(3), 187–196. https://doi.org/10.1260/0266-3511.28.3-4.187

Lienhard, J., & Knippers, J. (2015). Bending-Active Textile Hybrids. *Journal of the International Association for Shell and Spatial Structures*, *56*(1), 37–48.

Lienhard, J., Schleicher, S., Poppinga, S., Masselter, T., Milwich, M., Speck, T., & Knippers, J. (2011). Flectofin: A hingeless flapping mechanism inspired by nature. *Bioinspiration & Biomimetics*, *6*(4), 045001. https://doi.org/10.1088/1748-3182/6/4/045001

Liew, A., Van Mele, T., & Block, P. (2016). Vectorised graphics processing unit accelerated dynamic relaxation for bar and beam elements. *Structures*, *8*, 111–120. https://doi.org/10.1016/j.istruc.2016.09.002

Liu, Y., Pottmann, H., Wallner, J., Yang, Y.-L., & Wang, W. (2006). Geometric modeling with conical meshes and developable surfaces. *ACM Transactions on Graphics*, *25*(3), 681. https://doi.org/10.1145/1141911.1141941

Maclaurin, D., Duvenaud, D., Johnson, M., & Townsend, J. (2015). *Autograd: Efficiently computes derivatives of numpy code* [Python]. Harvard Intelligent Probabilistic Systems Group. https://github.com/HIPS/autograd (Original work published 2014)

Malerba, P. G., Patelli, M., & Quagliaroli, M. (2012). An Extended Force Density Method for the form finding of cable systems with new forms. *Structural Engineering and Mechanics*, *42*(2), 191–210. https://doi.org/10.12989/sem.2012.42.2.191

Maraniello, S., & Palacios, R. (2016). Optimal vibration control and co-design of very flexible actuated structures. *Journal of Sound and Vibration*, *377*, 1–21. https://doi.org/10.1016/j.jsv.2016.05.018

Marmo, F., Demartino, C., Candela, G., Sulpizio, C., Briseghella, B., Spagnuolo, R., Xiao, Y., Vanzi, I., & Rosati, L. (2019). On the form of the Musmeci's bridge over the Basento river. *Engineering Structures*, *191*, 658–673. https://doi.org/10.1016/j.engstruct.2019.04.069

Marquardt, D. (1963). An Algorithm for Least-squares Estimation of Nonlinear Parameters. *SIAM Journal Applied Mathematics*, *11*, 431–441.

Mayencourt, P. L., Giraldo, J. S., Wong, E., & Mueller, C. T. (2017). Computational Structural Optimization and Digital Fabrication of Timber Beams. In A. Bögle & M. Grohmann (Eds.), *Proceedings of the IASS Annual Symposium*.

Merrick, J. (2006). Glenn Howells / Savill Building. *Architect's Journal*, *224*(1), 25–39.

Mesnil, R., Douthe, C., Baverel, O., & Léger, B. (2016). Marionette Mesh: From descriptive geometry to fabrication-aware design. *Advances in Architectural Geometry 2016*, 204–221. https://doi.org/10.3218/3778-4

Mesnil, R., Ochsendorf, J., & Douthe, C. (2015). Stability of Pseudo-Funicular Elastic Grid Shells. *International Journal of Space Structures*, *30*(1), 27–36. https://doi.org/10.1260/0266-3511.30.1.27

Miki, M., & Kawaguchi, K. (2010). Extended force density method for form finding of tension structures. *Journal of the International Association for Shell and Spatial Structures*, *51*, 291–303.

Mork, H. J., Dyvik, H. S., Manum, B., Rønnquist, A., & Labonnote, N. (2016). Introducing the segment lath—A simplified modular timber gridshell built in Trondheim Norway. *World Conference on Timber Engineering*, 1–8.

Nabaei, S., Baverel, O., & Weinand, Y. (2013). Mechanical form-Finding of the Timber Fabric Structures with Dynamic Relaxation Method. *International Journal of Space Structures*, *28*(3–4), 197–214. https://doi.org/10.1260/0266-3511.28.3-4.197

Narain, R., Overby, M., & Brown, G. E. (2016). ADMM ⊇ Projective Dynamics: Fast Simulation of General Constitutive Models. In J. Jorge & M. Lin (Eds.), *ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

Nealen, A., Müller, M., Keiser, R., Boxerman, E., & Carlson, M. (2006). Physically Based Deformable Models in Computer Graphics. *Computer Graphics Forum*, *25*(4), 809–836. https://doi.org/10.1111/j.1467-8659.2006.01000.x

Nesterov, Y. (1983). A method of solving a convex programming problem with convergence rate O (1/k2). *Soviet Mathematics Doklady*, *27*(2).

Nicholas, P. (2013). Graded Territories. In *Designing Material Materialising Design* (pp. 49–68).

Nørgaard, S. A., Sagebaum, M., Gauger, N. R., & Lazarov, B. S. (2017). Applications of automatic differentiation in topology optimization. *Structural and Multidisciplinary Optimization*, *56*(5), 1135–1146. https://doi.org/10.1007/s00158-017-1708-2

Ohlbrock, P. O., & Schwartz, J. (2015, August 20). Combinatorial equilibrium modelling. *Future Visions - Graphic Computation*. IASS Annual Symposia, Amsterdam.

Olcayto, R. (2007, June 22). Solutions: Timber Structures—Gridshell glazes over the past. *Building Design*, *1776*, 14–17.

Orr, J. (2012). *Flexible formwork for visual concrete* [Ph.D. Dissertation]. University of Bath.

Panetta, J., Konaković-Luković, M., Isvoranu, F., Bouleau, E., & Pauly, M. (2019). X-Shells: A New Class of Deployable Beam Structures. *ACM Trans. Graph.*, *38*(4), 83:1–83:15. https://doi.org/10.1145/3306346.3323040

Panozzo, D., Block, P., & Sorkine-Hornung, O. (2013). Designing unreinforced masonry models. *ACM Transactions on Graphics*, *32*, 1. https://doi.org/10.1145/2461912.2461958

Paszke, A., Gross, S., Massa, F., Lerer, A., Bradbury, J., Chanan, G., Killeen, T., Lin, Z., Gimelshein, N., Antiga, L., Desmaison, A., Kopf, A., Yang, E., DeVito, Z., Raison, M., Tejani, A., Chilamkurthy, S., Steiner, B., Fang, L., … Chintala, S. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d\textquotesingle Alché-Buc, E. Fox, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 32* (pp. 8026–8037). Curran Associates, Inc. http://papers.nips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Pérez, J., Thomaszewski, B., Coros, S., Bickel, B., Canabal, J. A., Sumner, R., & Otaduy, M. A. (2015). Design and fabrication of flexible rod meshes. *ACM Transactions on Graphics*, *34*(4), 138:1–138:12. https://doi.org/10.1145/2766998

Petersen, K. B., & Pedersen, M. S. (2012). *The Matrix Cookbook* [Book manuscript]. http://matrixcookbook.com

Piker, D. (2016a). *Kangaroo 2 Goals*. https://github.com/Dan-Piker/K2Goals/tree/2a2daccbf63206755ad964c0b0c1a23ee18b0aa4

Piker, D. (2016b). *Kangaroo physics*. http://www.food4rhino.com/app/kangaroo-physics

Piker, D. (2017a). *Kangaroo solver*. http://www.grasshopper3d.com/xn/detail/2985220:Comment:1718553

Piker, D. (2017b). *Kangaroo solver*. Kangaroo Group on Grasshopper Forums. http://www.grasshopper3d.com/xn/detail/2985220:Comment:1718553

Pottmann, H., Eigensatz, M., Vaxman, A., & Wallner, J. (2015). Architectural geometry. *Computers and Graphics*, *47*, 145–164. https://doi.org/10.1016/j.cag.2014.11.002

Pottmann, H., Liu, Y., Wallner, J., Bobenko, A., & Wang, W. (2007). Geometry of multi-layer freeform structures for architecture. *ACM Transactions on Graphics*, *26*(3), 65. https://doi.org/10.1145/1276377.1276458

Quinn, G., Deleuran, A. H., Piker, D., Brandt-olsen, C., Tamke, M., Thomsen, M. R., & Gengnagel, C. (2016). Calibrated and Interactive Modelling of Form-Active Hybrid Structures. In K. Kawaguchi, M. Ohsaki, & T. Takeuchi (Eds.), *Spatial Structures: Proceedings of the IASS Symposium* (pp. 1–9).

Rao, C., Tian, L., Yan, D.-M., Liao, S., Deussen, O., & Lu, L. (2019). Consistently fitting orthopedic casts. *Computer Aided Geometric Design*, *71*, 130–141. https://doi.org/10.1016/j.cagd.2019.04.018

Rippmann, M., Lachauer, L., & Block, P. (2012). Interactive Vault Design. *International Journal of Space Structures*, *27*(4), 219–230. https://doi.org/10.1260/0266-3511.27.4.219

Rustamov, R. M., Ovsjanikov, M., Azencot, O., Ben-Chen, M., Chazal, F., & Guibas, L. (2013). Map-based exploration of intrinsic shape differences and variability. *ACM Transactions on Graphics*, *32*(4), 1. https://doi.org/10.1145/2461912.2461959

Sakai, Y., Ohsaki, M., & Adriaenssens, S. (2020). A 3-dimensional elastic beam model for form-finding of bending-active gridshells. *International Journal of Solids and Structures*, *193–194*, 328–337. https://doi.org/10.1016/j.ijsolstr.2020.02.034

Sánchez, J., Serna, M. Á., & Morer, P. (2007). A multi-step force-density method and surface-fitting approach for the preliminary shape design of tensile structures. *Engineering Structures*, *29*, 1966–1976. https://doi.org/10.1016/j.engstruct.2006.10.015

Sasaki, M. (2014). Structural design of free-curved RC shells. In *Shell Structures for Architecture: Form Finding and Optimization* (pp. 259–270). Routledge.

Schek, H.-J. (1974). The force density method for form finding and computation of general networks. *Computer Methods in Applied Mechanics and Engineering*, *3*, 115–134. https://doi.org/10.1016/0045-7825(74)90045-0

Sehlström, A., Isaksson, J., & Skeppstedt, M. (2018). Pre-stressed Geodesic Gridshell. *Posters of the Advances in Architectural Geometry Conference*. Advances in Architectural Geometry, Göteborg, Sweden. https://research.chalmers.se/en/publication/507931

Senatore, G., & Piker, D. (2014). Interactive real-time physics. *Computer-Aided Design*, *61*, 32–41. https://doi.org/10.1016/j.cad.2014.02.007

Skouras, M., Thomaszewski, B., Kaufmann, P., Garg, A., Bickel, B., Grinspun, E., & Gross, M. (2014). Designing Inflatable Structures. *ACM Transactions on Graphics*, *33*(4), 1–10. http://dx.doi.org/10.1145/2601097.2601166

Soriano, E., Sastre, R., & Boixader, D. (2019). G-shells: Flat collapsible geodesic mechanisms for gridshells. In C. Lázaro, K.-U. Bletzinger, & E. Oñate (Eds.), *Form and Force: Proceedings of the IASS Annual Symposium 2019 – Structural Membranes 2019*. International Association for Shell and Spatial Structures (IASS).

Tamke, M., Baranovskaya, Y., Deleuran, A. H., Monteiro, F., Fangueiro, R. M. E. S., Stranghöhner, N., Uhlemann, J., Schmeck, M., Gengnagel, C., & Thomsen, M. R. (2016). Bespoke Materials For Bespoke Textile Architecture. In K. Kawaguchi, M. Ohsaki, & T. Takeuchi (Eds.), *Spatial Structures: Proceedings of the IASS Symposium*.

Tang, C., Sun, X., Gomes, A., Wallner, J., & Pottmann, H. (2014). Form-finding with Polyhedral Meshes Made Simple. *ACM Trans. Graph.*, *33*(4), 70:1–70:9. https://doi.org/10.1145/2601097.2601213

The Mathworks Inc., a. (2015). *MATLAB and Optimization Toolbox Release 2015b*. The MathWorks, Inc.

Van Mele, T., & Block, P. (2011). A novel form finding method for fabric formwork for concrete shells. *Journal of the International Association for Shell and Spatial Structures*, *52*(170), 217–224.

Van Mele, T., De Laet, L., Veenendaal, D., Mollaert, M., Block, P., & Mele, V. (2013). Shaping Tension Structures with Actively Bent Linear Elements. *International Journal of Space Structures*, *28*(3–4), 127–135. https://doi.org/10.1260/0266-3511.28.3-4.127

Van Mele, T., Panozzo, D., Sorkine-Hornung, O., & Block, P. (2014). Best-fit thrust network analysis: Rationalization of freeform meshes. In *Shell Structures for Architecture: Form Finding and Optimization* (pp. 157–168). Routledge.

Veenendaal, D. (2017). *Design and form finding of flexibly formed shell structures* [PhD Thesis, ETH Zurich, Department of Architecture]. https://doi.org/10.3929/ethz-a-010831669

Veenendaal, D., & Block, P. (2012). An overview and comparison of structural form finding methods for general networks. *International Journal of Solids and Structures*, *49*(26), 3741–3753. https://doi.org/10.1016/j.ijsolstr.2012.08.008

Vouga, E., Mathias, H., & Wallner, J. (2012). Design of Self-supporting Surfaces. *ACM Transactions on Graphics (TOG)*, *31*(4), 87–97.

Wallner, J., & Pottmann, H. (2011). Geometric Computing for Freeform Architecture. *Journal of Mathematics in Industry*, *1*(1), 4. https://doi.org/10.1186/2190-5983-1-4

Wels, S. (2008). *California Academy of Sciences: Architecture in harmony with nature*. Chronicle Books.

Williams, C. (2014). What is a shell? In S. Adriaenssens, P. Block, D. Veenendaal, & C. Williams (Eds.), *Shell Structures for Architecture: Form Finding and Optimization* (pp. 21–31). Routledge.

Wu, J., Lu, E., Kohli, P., Freeman, B., & Tenenbaum, J. (2017). Learning to See Physics via Visual De-animation. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, & R. Garnett (Eds.), *Advances in Neural Information Processing Systems 30* (pp. 153–164). Curran Associates, Inc. http://papers.nips.cc/paper/6620-learning-to-see-physics-via-visual-de-animation.pdf

Zehnder, J., Coros, S., & Thomaszewski, B. (2016). Designing structurally-sound ornamental curve networks. *ACM Transactions on Graphics*, *35*(4), 99:1–99:10. https://doi.org/10.1145/2897824.2925888

Zhang, X., Fang, G., Skouras, M., Gieseler, G., Wang, C. C. L., & Whiting, E. (2019). Computational design of fabric formwork. *ACM Transactions on Graphics*, *38*(4), 109:1–109:13. https://doi.org/10.1145/3306346.3322988