

# **Computational Methods for Fabrication-aware Modeling, Rationalization and Assembly of Architectural Structures**

THÈSE N° 6685 (2015)

PRÉSENTÉE LE 7 SEPTEMBRE 2015

À LA FACULTÉ INFORMATIQUE ET COMMUNICATIONS  
LABORATOIRE D'INFORMATIQUE GRAPHIQUE ET GÉOMÉTRIQUE  
PROGRAMME DOCTORAL EN INFORMATIQUE ET COMMUNICATIONS

ÉCOLE POLYTECHNIQUE FÉDÉRALE DE LAUSANNE

POUR L'OBTENTION DU GRADE DE DOCTEUR ÈS SCIENCES

PAR

**Mario Moacir DEUSS**

acceptée sur proposition du jury:

Prof. S. Süssstrunk, présidente du jury  
Prof. M. Pauly, directeur de thèse  
Prof. M. Botsch, rapporteur  
Dr D. Panozzo, rapporteur  
Prof. Y. Weinand, rapporteur



ÉCOLE POLYTECHNIQUE  
FÉDÉRALE DE LAUSANNE

Suisse  
2015





It matters not how strait the gate,  
How charged with punishments the scroll,  
I am the master of my fate:  
I am the captain of my soul.  
— William Ernest Henley

To my family and friends.



# Acknowledgments

I would like to thank my mother, my father and my sister very much for their never ending support and unconditional love.

I would like to express my sincere gratitude to my advisor Mark Pauly. Your inspiring lectures on computer graphics and geometry during my studies at ETH Zürich not only provided me with the necessary theoretical and practical knowhow for my PhD but also deepened my interest in the subjects. Thank you also for supporting me with my Master Thesis at Stanford University. You were a great and kind mentor and teacher to me.

I would like to thank Yang Liu and Microsoft for my summer research internship in Beijing, China. Yang, you gave me all the freedom and support necessary to study the topic of self-supporting structures. Special thanks to all the national and international interns turning that internship into a fascinating cultural experience. Thank you Daniele Panozzo for the close collaboration and nurturing support during our project on self-supporting structures. Thank you Leonidas Guibas for supporting me with my Master Thesis at Stanford University.

I would like to thank Sabine Süsstrunk, Mario Botsch, Yves Weinand and Daniele Panozzo for being part of my thesis committee.

I would like to acknowledge my many collaborators without whom my PhD would not have been the same. Many thanks to Anders Holden Deleuran, Bailin Deng, Sofien Bouaziz, Daniel Piker and Mark Pauly for the collaboration on the ShapeOp project. Thank you, Michael Eigensatz, for introducing into the world of architectural geometry and your software tool.

Also I thank Alexander Schiftner, Helmut Pottmann, Johannes Wallner, Niloy Mitra and Mark Pauly for the collaboration on our publications of cost-optimized paneling. In the context of this project, I would like to thank Yves Brise, Peter Kaufmann and Sebastian Martin for their help. Special thanks to Formtexx for providing the architectural datasets and to RFR for fruitful comments.

Thanks to Daniele Panozzo, Emily Whiting, Yang Liu, Philippe Block, Olga Sorkine-Hornung and Mark Pauly for the collaboration on our project on assembling self-supporting structures. In the context of this project I would like to thank Hao Pan and

---

Xiaoming Fu for providing their source code and support, and Etienne Vouga, Fernando de Goes, Ramon Weber and Matthias Rippmann for providing datasets, as well as Bailin Deng, Andrea Tagliasacchi and Sofien Bouaziz for inspiring discussions.

Many thanks to all the members of LGG for your inspiration, support, co-teaching, knowledge-sharing and feedback. Thank you also for all the fun times we had together. Thank you Sofien Bouaziz for being a never ending source of inspiration and motivation for research. Thank you Bailin Deng for your patient supervision. Thank you Duygu Ceylan for being there early on and sharing our office with me. Thank you Andrea Tagliasacchi for your competent advice and interesting discussions. Thank you Juyong Zhang for collaboration and your kind invitation to University of Science and Technology of China. Thank you Yuliy Schwartzburg for collaboration and proofreading. Many thanks to Boris Neubert, Thibaut Weise, Hao Li, Mina Konakovic, Anastasia Tkach, Alexandru-Eugen Ichim, Stefan Lienhard, Minh Dang, Romain Testuz for making the LGG such a unique lab.

Thank you very much Madeleine Robert for your support in any kind of situation during my PhD. May your laughter keep the aisles of BC filled with joy. I would also like to thank our visitors Keenan Crane and Justin Solomon for interesting lectures and discussions, our summer interns Laura Gosmino, Ian Dewancker, Mihita Cvitanović and Rosália Schneider for the collaboration and the students I supervised for their trust in me.

Special thanks to my friends in Lausanne, Zürich and all around the world for all the good times and support during my PhD, to Capoeira ACL and Capoeira CTE for the refreshing trainings and events, and to the MADdancers for the great practices, performances and tasty dinners.

My research was supported by the SNF Grant (200021-137626) and received funding from the European Research Council under the European Union's 7th Framework Programme/ERC Grant Agreement 257453, ERC Starting Grant COSYM.

*Lausanne, 9 July 2015*

Mario Deuss

# Abstract

Architectural structures such as buildings, towers, bridges and roofs are of fundamental importance in urban environments. Their design, planing, construction and maintenance carry numerous challenges in engineering due to the complex interplay of material, form, spaces and statics. The history of architecture shows proof of the many approaches humanity has come up with to tackle those challenges. To mention a few, think of tents, timber huts, pyramids, masonry bridges, steel structures and modern skyscrapers. The methods presented in this thesis are tailored to handle the general case of freeform architectural structures.

A common way to develop novel architectural structures is to produce prototypes at various scales. Physical prototypes, however, do not allow for quick changes of aspects of a design such as material and form. The advent of computer-aided design tools alleviated some of these limitations, but brought with it new challenges in terms of simulation of physics and interaction with virtual content. Compared to physical prototypes, digital ones can in theory include many more types of constraints by leveraging numerical computation. To be practical, however, a digital prototyping tool needs to be designed carefully considering efficiency, generality, accuracy, simplicity and robustness of its implementation. Also, there are currently many unsolved problems in the digital exploration of desirable and feasible designs with respect to constraints.

Important constraints are imposed directly or indirectly by the ease and cost of realization and maintenance of a freeform architectural structure. For example, the geometry of components making up a structure can have a big impact on the cost of fabrication: planar components can simply be cut out of material that usually comes in flat sheets, while curved ones tend to require more costly production processes. The construction and assembly of architectural structures can cause a considerable part of the full cost. A well-chosen assembly sequence can reduce both labor and necessary temporary support structures such as scaffolds.

This thesis approaches challenges and open questions in the context of computational prototyping tools for architectural structures by studying three concrete subproblems and proposing practical solutions to them.

We present a constraint-aware modeling tool capable of robustly simulating physics

---

and handling various geometric constraints at interactive rates. We show how constraints relevant to aesthetics and production can be implemented in our tool, yielding a fabrication-aware modeling tool.

We present a computational method for finding a mass-producible approximation of a given surface which minimizes fabrication cost. The method optimizes for a set of molds each of which can be used to produce multiple components, so-called panels, of the approximation, while respecting user-defined constraints on the continuity and deviation from the input surface. The problem this method solves is referred to as paneling, which in turn is an instance of the rationalization problem: approximating input under constraints relevant to the physical realization of a structure.

We present a computational method for minimizing the work necessary for the construction of a freeform self-supporting structure. We study the use of chains to support the structure during assembly. Our method searches for an assembly sequence of the structure's components which minimizes the number of times a chain has to be rehung.

Key words: architectural geometry, complex assembly, constrained optimization, paneling, rationalization, fabrication-aware modeling, self-supporting structures

# Zusammenfassung

Architektonische Strukturen sowie Gebäude, Türme, Brücken und Überdachungen sind fundamental wichtig in urbanen Umgebungen. Design, Planung, Konstruktion und Wartung bergen zahlreiche technische Herausforderungen aufgrund des komplexen Zusammenspiels von Material, Form, Räumen und Statik. Die Geschichte der Architektur zeigt eine Vielzahl von Angehenweisen der Menschheit, um diese Herausforderungen in Angriff zu nehmen. Man denke zum Beispiel an Zelte, Holzhütten, Pyramiden, Steinbrücken, Stahlstrukturen und moderne Wolkenkratzer. Die Methoden, welche in dieser Dissertation präsentiert werden, sind für den allgemeinen Fall der architektonischen Freiformstrukturen ausgelegt.

Eine verbreitete Methode zur Entwicklung neuartiger architektonischer Strukturen ist der Gebrauch von Prototypen verschiedener Grössenordnungen. Physikalische Prototypen aber beschränkten bisher die Möglichkeit Aspekte des Designs wie Material und Form rasch zu ändern. Die Entwicklung von computer-assistierten Design-Programmen half diese Beschränkungen aufzuheben, brachten aber neue Herausforderungen bezüglich der Physiksimulation und Interaktion mit virtuellem Inhalt. Im Vergleich zu physikalischen können digitale Prototypen theoretisch eine Vielzahl andersartiger Bedingungen mit Hilfe von numerischen Berechnungen miteinbeziehen. Um von praktischer Relevanz zu sein, sollte ein Programm für digitale Prototypen unter sorgfältiger Berücksichtigung von Effizienz, Generalität, Präzision, Einfachheit und Robustheit entwickelt werden. Auch die rechnergestützte Erkundung von gewünschten und realisierbaren Designs unter Bedingungen birgt viele ungelöste Fragestellungen.

Wichtige Bedingungen leiten sich direkt oder indirekt von den Realisierungs- und Wartungskosten einer architektonischen Freiformstruktur her. Die Form der Komponenten einer Struktur zum Beispiel kann eine grosse Auswirkung auf die Fabrikationskosten haben: Planare Komponenten kann man ohne grossen Aufwand aus Materialien, die normalerweise in flachen Platten geliefert werden, ausschneiden, währenddessen gekrümmte Komponenten tendenziell kostenintensivere Produktionsmethoden verlangen. Die Konstruktion und Montage von architektonischen Strukturen kann auch einen erheblichen Anteil der Gesamtkosten verursachen. Eine gut gewählte Montageabfolge kann sowohl den Arbeitsaufwand wie auch die benötigten Baustützen wie zum Beispiel Gerüste erheblich reduzieren.

---

Diese Dissertation geht die Herausforderungen und offenen Fragen im Kontext der rechnergestützten Programme für digitale Prototypen von architektonischen Strukturen an, indem sie drei konkrete Teilprobleme studiert und dazu praktische Lösungen vorschlägt.

Wir präsentieren ein Modellierungsprogramm, das instande ist in interaktivem Tempo physikalische Aspekte unter Einbezug einer Vielzahl von geometrischen Bedingungen robust zu simulieren. Wir zeigen wie Bedingungen bezüglich Ästhetik und Produktion in unserem Programm implementiert werden können.

Wir präsentieren eine rechnergestützte Methode zur Berechnung einer massenproduzierbaren Annäherung einer gegebenen Oberfläche, welche Fabrikationskosten minimiert. Die Methode berechnet Press- und Gussformen, mit welchen mehrere Komponenten der Annäherung hergestellt werden können. Gleichzeitig respektiert die Methode benutzerdefinierte Bedingungen an die Kontinuität und Abweichung von der gegebenen Oberfläche. Das damit gelöste Problem ist eine Instanz des Rationalisierungsproblems: Die Annäherung eines digitalen Designs an die physikalische Realität einer Struktur.

Wir präsentieren eine rechnergestützte Methode zur Minimierung des Arbeitsaufwandes, der für den Bau einer selbststützenden Freiformstruktur nötig ist. Wir untersuchen die Verwendung von Ketten, um die Struktur während des Baus zu stützen. Unsere Methode sucht eine Montagereihenfolge für die Komponenten der Struktur, welche die Anzahl der nötigen Kettenneuplazierungen minimiert.

Stichwörter: Architekturgeometrie, komplexe Montagereihenfolgen, bedingte rechnergestützte Optimierung, Fabrikation, Modellierung, Panelisierung, Rationalisierung, selbststützende Strukturen



# Contents

<b>Acknowledgments</b>	<b>i</b>
<b>Abstract (English/Deutsch)</b>	<b>iii</b>
<b>List of Figures</b>	<b>xi</b>
<b>List of Tables</b>	<b>xix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Architectural Structures . . . . .	1
1.1.1 Challenges . . . . .	2
1.2 Modeling . . . . .	3
1.2.1 Prototyping . . . . .	3
1.2.2 Properties of Digital Prototyping Tools . . . . .	3
1.2.3 Constrained Modeling . . . . .	4
1.2.4 Combinatorial and Continuous Modeling . . . . .	4
1.2.5 Hard and Soft Constraints . . . . .	5
1.2.6 Rationalization and Paneling . . . . .	5
1.2.7 Constraint-aware Modeling . . . . .	6
1.2.8 Design Spaces . . . . .	6
1.3 Fabrication Constraints . . . . .	6
1.3.1 Cutting . . . . .	7
1.3.2 Milling . . . . .	7
1.3.3 Bending and Folding . . . . .	7
1.3.4 Casting and Molds . . . . .	8
1.3.5 3D Printing . . . . .	8
1.4 Assembly . . . . .	8
1.4.1 Joints . . . . .	9
1.5 Computational Tools . . . . .	9
1.6 Contributions . . . . .	10
1.7 Publications . . . . .	10
<b>2 Related Work</b>	<b>13</b>

## Contents

---

2.1	Constrained Modeling . . . . .	13
2.1.1	Combinatorial Modeling . . . . .	14
2.1.2	Paneling . . . . .	14
2.1.3	Rationalization . . . . .	16
2.1.4	Self-supporting Structures . . . . .	16
2.1.5	Constrained Numerical Optimization . . . . .	16
2.2	Assembly . . . . .	18
2.2.1	Masonry Building Fabrication . . . . .	18
2.2.2	Optimizing Construction Sequences . . . . .	18
2.2.3	3D Printing . . . . .	19
2.2.4	Rationalization and Assembly . . . . .	19
<b>3</b>	<b>ShapeOp</b>	<b>21</b>
3.1	Introduction . . . . .	21
3.2	Solver . . . . .	22
3.2.1	Proximity Function . . . . .	23
3.2.2	Shape Proximity for Geometric Data . . . . .	25
3.3	Projections . . . . .	27
3.3.1	Closeness . . . . .	28
3.3.2	Orientation . . . . .	28
3.3.3	Continuous Shapes . . . . .	29
3.3.4	Relative Shapes . . . . .	29
3.3.5	Polygonal Shapes . . . . .	30
3.4	Implementation . . . . .	34
3.5	Examples . . . . .	35
3.6	Design Process . . . . .	40
<b>4</b>	<b>Cost-Optimized Paneling of Architectural Freeform Surfaces</b>	<b>41</b>
4.1	Introduction . . . . .	41
4.2	Panels and Fabrication . . . . .	46
4.3	Paneling Architectural Freeform Surfaces . . . . .	48
4.3.1	Problem Specification . . . . .	48
4.3.2	Paneling Algorithm . . . . .	49
4.4	Formalization of the Paneling Algorithm . . . . .	52
4.4.1	Problem . . . . .	52
4.4.2	Algorithm . . . . .	53
4.4.3	Continuous Optimization . . . . .	53
4.4.4	Discrete Optimization . . . . .	55
4.4.5	Alternating Optimization . . . . .	56
4.5	Extensions . . . . .	57
4.5.1	Sharp Features . . . . .	57
4.5.2	Adaptive Control of Paneling Quality . . . . .	57
4.6	Case Studies . . . . .	58

4.6.1	Facade Design Study . . . . .	58
4.6.2	Skipper Library . . . . .	60
4.6.3	Lissajous Tower . . . . .	60
<b>5</b>	<b>Assembling Self-Supporting Structures</b>	<b>67</b>
5.1	Introduction . . . . .	68
5.2	Method . . . . .	71
5.2.1	Construction State Validity . . . . .	72
5.2.2	Static Equilibrium . . . . .	73
5.2.3	Global Equilibrium Analysis . . . . .	74
5.2.4	Construction Sequence . . . . .	78
5.2.5	Practical Constraints . . . . .	79
5.2.6	Manufacturing Tolerances and Safety Factors . . . . .	81
5.3	Results . . . . .	81
5.3.1	Large-scale Simulations . . . . .	82
5.3.2	Self-supporting Puzzles . . . . .	82
5.3.3	Validation via Small-scale Models . . . . .	85
5.3.4	Restoration of Historical Buildings . . . . .	85
5.3.5	Evaluation of Safety Factors . . . . .	86
5.4	Implementation . . . . .	86
5.4.1	$L_p$ -Relaxation . . . . .	86
5.4.2	Iterative Reweighting . . . . .	86
5.4.3	Friction Cones . . . . .	87
<b>6</b>	<b>Discussion</b>	<b>89</b>
6.1	ShapeOp . . . . .	89
6.2	Paneling . . . . .	90
6.3	Assembly . . . . .	91
6.4	Implicit and Explicit Constraints . . . . .	93
6.5	Future Work . . . . .	94
	<b>Bibliography</b>	<b>103</b>
	<b>Curriculum Vitae</b>	<b>105</b>



# List of Figures

1.1	Examples of freeform architectural structures. From left to right, top row: Yas Marina, Abu Dhabi by Stephen Colebourne, CC BY 2.0. Harbour Bridge, Sdney by helen@littlethorpe, CC BY NC ND 2.0. Centre Pompidou, Metz by Jean-Pierre Dalbéra, CC BY 2.0. Bottom row: Stone Bridge, Pakenham by Shawn Nystrand, CC BY SA 2.0. British Museum, London by Dave Catchpole, CC BY 2.0. Hungerburgbahn, Innsbruck by IngolfBLN, CC BY SA 2.0 . . . . .	1
1.2	Various designs of a design space defined by a hard, co-circular constraint for each grid line, and soft squareness objective per quad. Figure from [BDS <sup>+</sup> 12]. . . . .	6
3.1	A quad mesh constrained to consist of squares illustrating the ShapeOp solver. Left: Initial configuration. Middle: Local step - Projecting each quad onto its closest square. Right: Global step - Joining the individual projections by a global minimization. The resulting mesh is then used as initial configuration and the solver iterates. . . . .	24
3.2	The proximity function $\phi(\mathbf{x})$ is the weighted sum of squared distances $d_i(\mathbf{x})$ of the point $\mathbf{x}$ to the projections $P_i(\mathbf{x})$ onto the respective constraint sets $C_i$ . Minimizing $\phi(\mathbf{x})$ yields a feasible solution if the constraint sets intersect (left), and a least-squares solution otherwise (right). . . . .	25
3.3	Two iterations of the two-step minimization of the proximity function $\phi(\mathbf{x})$ with $w_i = 1$ . Step I computes the projections using the current estimate $\mathbf{x}$ . Step II updates $\mathbf{x}$ by minimizing $\phi(\mathbf{x})$ keeping the projections fixed. At each step, $\phi(\mathbf{x})$ , illustrated by the sum of the error bars, will decrease, even if some of the individual elements increase. . . . .	26
3.4	Our optimization alternates between projection and linear solve. In this example, we prescribe a regular polygon constraint that pushes all quadrilaterals to become squares. The projection finds the best matching square for each quadrilateral to determine the target position for each vertex. The linear solve reconciles these projected positions in a least-squares sense. . . . .	27
3.5	Schematic overview of our implementation of ShapeOp. . . . .	34

3.6	The Grasshopper definition used for the hanging cloth example seen in Fig. 3.7. . . . .	35
3.7	Use of ShapeOp for physics simulation of elastic materials. A hanging cloth modelled using edge strain and bending constraints. The three vertices are anchored using closeness constraints and all points are subjected to a gravity load. Left: The input mesh. Middle: The constrained mesh at the first solve iteration in which the anchors are immediately moved very far apart. Right: The constrained mesh after 100 iterations with the anchor point moved back to their starting positions. Top: Wireframe rendering with the edges coloured by their strain (red = high, blue = low). Bottom: Shaded rendering. The example demonstrates both the stability and the fast convergence of the solver. . . . .	36
3.8	Use of ShapeOp for constrained modeling of a shell with rational geometric properties. The vertices on the parameter lines of a quad-mesh are constrained to always lie on a circular arc using the circle constraint. Each face is constrained towards being square using the similarity constraint. Five vertices are anchored to different positions than their initial positions, enabling shape exploration. Left to right: 1) The input mesh, the face used for similarity and vectors visualizing start/end positions for the anchors. 2) The constrained mesh after 10 iterations. 3) The constrained mesh after 300 iterations. 4) The constrained mesh with circles drawn through each of the parameter line vertices (Red = Line vertices distance to circle is large, Blue = Line vertices distance to circle is small). . . . .	36
3.9	Use of ShapeOp for rationalizing an existing geometry. Each face of the quad-mesh is constrained towards being planar using the plane constraint. Each vertex is constrained to its initial position using the closeness constraint by a small weight to maintain the shape of the mesh. Left: Input mesh. Middle: The constrained mesh after 10 iterations. Right: The constrained mesh after 200 iterations. Top: Shaded rendering. Bottom: Planarity analysis rendering (Red = Low planarity, Green = High planarity). . . . .	37
3.10	Use of ShapeOp for constrained modeling of box shape with multiple rigid shape targets. A quad-mesh box is anchored at the vertices on two sides of the box. The image sequence shows the vertices on one side being pulled away over time. As this occurs each mesh face attempts to project itself onto one of the three shape targets below the box. The solver has been initialized using dynamics leading to the rippling effect as the faces switch their projection targets from short to medium to long. This projection type is enabled using the rigid constraint. . . . .	37

3.11	Use of ShapeOp for constrained modeling of a shell with topologically different shape targets. A planar mesh composed of both triangles and quads is anchored at four vertices using the closeness constraint. Using the similarity constraint, each face is constrained towards their initial shape i.e. an equilateral triangle or a square. 1) The input mesh. 2) The mesh after 1 iteration. 3) After 100 iterations. 4) After 500 iterations. . . . .	38
3.12	Use of ShapeOp for constrained modeling of a randomly generated quad-mesh with multiple constraints as design drivers. The example demonstrates the effect of applying the same constraints on meshes with different resolutions. It uses three primary constraints: 1) Limit the internal angles of each face to be within 80 and 110 degrees, 2) The boundaries of the mesh should lie on circles, 3) Each face should preserve its area. Additionally, a Laplacian of displacement constraint is added which smoothens out the mesh while maintaining the shape, and a bending constraint is added which ensures that face-face angles do not become too acute. The color code is a based on scoring system: The internal angles for each face are calculated. If an angle is within the desirable range it is scored 0, else 1. The scores are added for each face, best face score is 0 (Dark green) worst is 4 (Dark red). . . . .	38
3.13	Use of ShapeOp for funicular form finding. A hexagonal quad-mesh is anchored at each corner and subjected to an inverse gravity load. In this image sequence the only other constraint is that each edge should be 2.0 units long. This is implemented using the edge strain constraint. 1) The input mesh. 2) The constrained mesh after 1 iteration. 3) The constrained mesh after 10 iterations. 4) The constrained mesh after reaching equilibrium at iteration 1000. . . . .	39
3.14	Use of ShapeOp for funicular form finding under fabrication constraints. Demonstrates the effect of combining different constraints: Desired edge length, planarity of faces and desired range of internal face angles. All images show the constrained mesh at equilibrium. From left to right: 1) Shaded rendering. 2) Edge length deviation from desired length. 3) Face angles deviation from desired angle range. 4) Face Planarity Deviation (Red = High deviation, Green = Low deviation). Row 1: The mesh with edge strain constraints. Row 2: The mesh with edge strain and internal mesh face angles constraints. Row 3: The mesh with edge strain and face planarity constraints. Row 4: The mesh with edge strain, internal mesh face angles and face planarity constraints. . . . .	39

4.1	Given a reference surface (top row), the <i>paneling algorithm</i> produces a rationalization of the the input. The paneling solution (middle row) employs a small set of molds that can be reused for cost-effective panel production (bottom row), while preserving surface smoothness and respecting the original design intent. The shown metal paneling solution is 40% cheaper than the production alternative of using custom molds for each individual panel. Figure 4.11 presents a variety of solutions that achieve cost savings of up to 60%. Figure 4.4 lists the metal cost ratios used. . . . .	42
4.2	Projects involving double-curved panels where a separate mold has been built for each panel. These examples illustrate the importance of the curve network and the existing difficulties in producing architectural freeform structures. (Left: Peter Cook and Colin Fournier, Kunsthaus, Graz. Right: Zaha Hadid Architects, Hungerburgbahn, Innsbruck.) Figure taken from [EKS <sup>+</sup> 10]. . . . .	43
4.3	Comparison with other rationalization algorithms on a freeform facade design study. (a, b) Rationalization using a planar quad mesh and developable surface strips, respectively. (c-f) Rationalization using the paneling algorithm with 1° and 1/4° kink angle thresholds, shown along with visualization of respective mold types (using glass cost ratios listed in Figure 4.4). A detailed overview of mold reuse for (e) is shown in Figure 4.5. . . . .	45
4.4	The panel types currently supported by our algorithm and two typical cost sets. . . . .	46
4.5	Illustration of the mold depot and the cost model by means of the example shown in Figure 4.3(e). The colors of panels are saturated according to mold reuse. Figure 4.4 lists the glass cost ratios used for this example. . . . .	47
4.6	Terminology and variables used in the paneling algorithm. The reference surface $F$ and the initial curve network $\mathcal{C}$ are given as part of the design specification. The optimization solves for the mold depot $\mathcal{M}$ , the panel-mold assignment function $A$ , the shape parameters of the molds, the alignment transformations $T_i$ , and the curve network samples $\mathbf{c}_k$ . Figure taken from [EKS <sup>+</sup> 10]. . . . .	48
4.7	Example of mold reuse. Panel boundary curves are in general not congruent. However, several panels may be closely grouped together on the same mold base surface. In that case the same mold or machine configuration, which embraces all affected panels, may be used to manufacture the panels. This figure further illustrates how the congruent profiles of a rotational or translational surface, in this case the circles generating a torus, can be exploited for mold fabrication. . . . .	49



4.8	Illustrative comparison of different techniques for mold reuse. The curve should be approximated with circle arcs of varying radii. This can be understood as a simple paneling with cylinders of varying radii, where the figure shows an orthogonal cross section. The input design curve shown in <b>(a)</b> consists of nicely aligned circle arcs with decreasing radii from 25 to 5. The method shown in <b>(b)</b> clusters these radii (using k-means clustering) to obtain 3 molds and assigns the best mold to each segment. The colors indicate the segments sharing the same mold. The method shown in <b>(c)</b> does the same, but performs a clustering of $(1/\text{radius})$ instead of clustering the radius itself, which is a much better distance approximation for cylinders as shown in [EKS <sup>+</sup> 10] and therefore the maximal kink angle is already much lower compared to <b>(b)</b> . The method shown in <b>(d)</b> performs the full discrete optimization presented in [EKS <sup>+</sup> 10] and leads to an even better mold depot that enables a paneling with only 3 molds but very low kink angles. The differences presented on this schematic example become even more prominent if more complex surfaces and/or panel types are involved. . . . .	51
4.9	The paneling algorithm restricted to cylindrical panels. Here we compare a result on the Facade Design Study computed using simple local fitting of cylinders (a) to a paneling solution using only cylinders (b). For both results we show the axis directions of cylinders colored in magenta and the cumulative histograms of resulting divergences and kink angles. . . .	59
4.10	Adaptive quality control. . . . .	61
4.11	Effect of global vs spatially varying kink angle specifications on the Skipper Library dataset. Paneling solutions using a global kink angle specification (a) and using adaptive kink angle thresholds computed based on the extent of visibility while moving along the indicated ground paths (b, c). Left column images show the reflection lines on paneled surfaces, while right column images show the mold types for individual panels (color convention same as in Figure 4.1). Figures 4.12 and 4.13 show the same solutions from two other views. Figure 4.10 shows the spatially varying kink angle thresholds used in (b) and (c). . . . .	62
4.12	Effect of global vs spatially varying kink angle specifications on the Skipper Library dataset, along with statistics for corresponding paneling solutions (see also Figure 4.11). . . . .	63
4.13	Effect of global vs spatially varying kink angle specifications on the Skipper Library dataset. Please refer to Figure 4.11 for details. . . . .	64
4.14	Paneling solution respecting crease line(s) on the input model. The characteristic sharp feature line of the Lissajous Tower is preserved in our paneling solution. . . . .	65

## List of Figures

---

5.1	We propose a construction method for self-supporting structures that uses chains, instead of a dense formwork, to support the blocks during the intermediate construction stages. Our algorithm finds a work-minimizing sequence that guides the construction of the structure, indicating which chains are necessary to guarantee stability at each step. From left to right: a self-supporting structure, an intermediate construction stage with dense formwork, an intermediate construction stage with our method and the assembled model. . . . .	67
5.2	The Arch-Lock system [Dre13] is used to construct a simple arch (left), and a barrel vault (right). [Copyright photographs: Lock-Block Ltd. 2013]	69
5.3	Two intermediate construction stages of our optimized sequence (right) and a trivial z-ordering (left). Our sequence needs considerably less work (0.62 instead of 1.13 chain changes per state in average), while computing it takes 3.5 times longer than determining the sparse set of chains for the trivial sequence. . . . .	70
5.4	Our algorithm converts an input masonry model in a work-minimizing construction sequence. From left to right: Forces resulting from our global equilibrium analysis, arch-blocks as extracted from flood-fill, four different states of the construction sequence. In all our figures, the blocks and chains are color-coded as follows: blue for support, light yellow for free blocks, gold for the newly added block and chains, dashed lines for chains that can be removed, and black lines for other active chains. We also color-code the candidate blocks considered by our sequence optimization using the minimum of Equation (5.5) relative to the other candidates in the color-bar on the right. Candidate blocks leading to an invalid state are shown in black. . . . .	71
5.5	Construction site mockup. . . . .	72
5.6	Model of chain forces and contact forces at interfaces between blocks. . .	73
5.7	Global analysis on a synthetic example. Force vectors (red lines) indicate the magnitude of forces between blocks. . . . .	76
5.8	The parameter $\lambda$ controls the tradeoff between stable regions and introduced chains. Chain forces are shown as blue lines. From left to right, $\lambda = 0.15, 0.12, 0.06$ . . . . .	76
5.9	Extracting quasi-arches. This figure shows the quasi-arches extracted with two thresholds on the maximal normal component of interface forces. . . .	77
5.10	At each construction step we test the work cost for inserting an additional block and we select the one with minimal energy. We highlight in black that blocks for which the optimization failed to find a force distribution that satisfies the equilibrium constraints. . . . .	79

5.11	Adding a bound on the maximal force that an anchor can support generates a sequence that distributes the chain forces more evenly by adding additional chains. Left: Unbounded solution using 3 chains with max. $f_c = 7.1$ and max. anchor bound 7.1. Note the max. chain is the only one connected to the max. anchor. Right: Solution with 8 chains bounded by 3.5 and anchors bounded by 5. For details please refer to Equations (5.8) and (5.9). . . . .	80
5.12	A large concert hall designed with RhinoVault is constructed using a sequence generated by our algorithm. The entire structure is made of hexagonal blocks placed using an interleaved layout typical of masonry constructions. . . . .	80
5.13	A freeform self-supporting structure designed with [VHWP12]. We tessellated the surface with quadrilateral elements, which have no interlocking and are thus prone to sliding failures, requiring a considerable number of supporting chains to be stable in all intermediate stages. . . . .	81
5.14	Our algorithm can be used to design challenging physical puzzles. . . . .	83
5.15	We validate our algorithm by constructing a masonry structure using our optimized work-minimizing construction sequence. . . . .	84
5.16	Our algorithm finds a sparse set of chains that guarantee stability even after the removal of a subset of the blocks. This can be applied to restoration of masonry structures. . . . .	85
5.17	This quasi-arch needs a chain to be stable. When the chain is loosened, the arch collapses due to a torque failure. . . . .	86

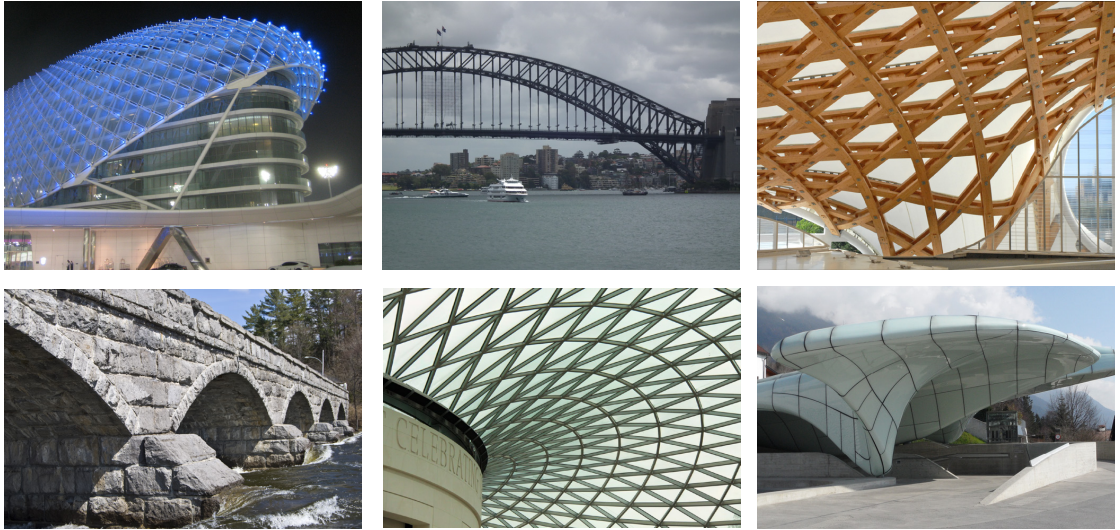


# List of Tables

3.1	Constraints implemented in ShapeOp. . . . .	33
4.1	Classification of panel types and typical production processes for common materials in architecture. Although we do not cover all the relevant production processes, this table is for a rough guideline. Planar panels have been left out. . . . .	44
5.1	Table with statistics for our results. From left to right: number of blocks, number of chains in the optimization, number of anchors, threshold parameter for the quasi-arch extraction, average number of used chains, average number of chain changes and computation time in minutes. . . . .	82
5.2	Average number of chains and chain changes for sequences of the model in Figure 5.1 computed with different parameter $p$ . The sequences were computed without quasi-arches to isolate the influence of the parameter. Note that $p = 1$ requires no reweighting. . . . .	87



# 1 Introduction



**Figure 1.1** – *Examples of freeform architectural structures. From left to right, top row: Yas Marina, Abu Dhabi by Stephen Colebourne, CC BY 2.0. Harbour Bridge, Sydney by helen@littlethorpe, CC BY NC ND 2.0. Centre Pompidou, Metz by Jean-Pierre Dalbéra, CC BY 2.0. Bottom row: Stone Bridge, Pakenham by Shawn Nystrand, CC BY SA 2.0. British Museum, London by Dave Catchpole, CC BY 2.0. Hungerburgbahn, Innsbruck by IngolfBLN, CC BY SA 2.0*

## 1.1 Architectural Structures

This thesis presents computational methods for fabrication-aware modeling, rationalization, and assembly of architectural structures such as buildings, roofs, and bridges. Furthermore, our methods are tailored to freeform structures of varying curvature, in contrast to traditional horizontal and vertical constructions. See Figure 1.1 for examples.

Large structures are often technically impossible or overly expensive to build in one piece. They are therefore commonly realized by assembling multiple components. There are also many advantages to subdividing a structure. Individual components can be fabricated independently and more cheaply using general-purpose machinery. Transporting components is easier than transporting the whole structure. The components can therefore be produced remotely, wherever most suitable, even in multiple places at once. Later in the life cycle of a structure, individual components can be replaced when necessary or reused in other structures.

### 1.1.1 Challenges

The components of a structure need to fit together with a certain tolerance depending on how they will be connected to each other. For example, glass panels mounted on a steel structure do not need to fit together as precisely as masonry blocks or bolted metal beams. Designing such structures is challenging—constraints, e.g. those dictated by the design intent, fabrication and assembly, make any change in a single component potentially propagate through the whole structure, turning a local edit into a global change. This renders manual edits extremely time-consuming, often infeasible in realistic budgets.

Computers however can potentially represent and edit thousands of components of an architectural structure at interactive rates. They are therefore an ideal complement to a designer’s creativity in our context. This thesis was developed in the context of *computer graphics*. We believe that computer graphics with its rich history in digital 3D modeling is a promising field of research to tackle the interdisciplinary challenges of computer-aided methods for architectural structures. In fact, a part of the computer graphics community began to study possibilities of incorporating architectural fabrication into 3D modeling under the topic of *architectural geometry* around the year of 2005 [PEVW15] in close collaboration with geometric mathematicians.

Major cost factors of large-scale structures are fabrication and assembly. Computation can greatly help to reduce fabrication cost, e.g. by forcing various components to be of the same shape. These components can then be mass-produced using the same machinery and settings, saving time and cost. An assembly order that reduces work and support material can further reduce cost considerably. Finding a mass-producible approximation of a structure and cost-minimizing assembly order are both globally coupled problems and therefore well-suited to be tackled, or at least assisted, by computers. Due to the lack of appropriate digital tools however, architectural structures are still developed in costly iterations between designers and technical experts. This thesis proposes computational tools to aid the user and minimize the iterations necessary between collaborators.



## 1.2 Modeling

In this thesis, we refer to *modeling* as the process of defining the geometry of a 3D architectural structure. Besides aesthetics and stability, we consider other criteria like cost of fabrication and assembly, which have an important practical influence on the modeling process. For the process of mathematically describing a problem, often referred to as mathematical modeling, we use the term *formalization*.

### 1.2.1 Prototyping

Traditionally, designs of architectural structures were often developed on scaled physical prototypes or on intuition gained thereof. While prototypes have the advantage of being tactile and highly responsive, many physical properties are not independent of scale and therefore, not well represented in a scaled prototype. Also, certain aspects such as material and geometry can be laborious to change. The advent of digital design tools has enabled designers to quickly vary appearance and adapt shape. Purely digital content like movie sets and characters or special effects have benefited tremendously from those tools.

When used for prototyping real world objects a digital design tool needs to simulate physical behavior. But accurate physical simulations are notoriously computationally intensive and limit the response time of the tool. Challenging designs are thus often first modeled geometrically while neglecting physics. Only later does an expert get to evaluate the feasibility of the model using expert knowledge and computationally intricate software tools. If any aspect of the model is infeasible, the designer needs to alter the model. Due to a lack of close collaboration with the experts, the designer can often only guess which modifications render the model feasible, which can lead to many costly iterations.

In response, researchers developed computational tools that use specific aspects of a design problem to overcome the limitations of more general physical simulations. Those tools typically not only evaluate the model, but additionally support the user by proposing changes that improve the design. While such a tool can effectively reduce or even replace the designer's interaction with an expert, it needs to be developed or adapted by computer scientists in close collaboration with experts separately for each project.

### 1.2.2 Properties of Digital Prototyping Tools

One can identify the following desirable properties of a digital prototyping tool:

1. Efficiency: To give the user an immersive experience comparable to the interaction with a physical prototype, a tool needs to reflect results as fast as possible.

2. Generality: The more individual aspects a tool can cope with in an integrated fashion, the more expressive a user can be, and the more novel are resulting designs. Additionally, time-consuming and costly iterations between tools and experts can be reduced.
3. Accuracy: A digital prototyping tool should accurately reflect the behavior of a physical full-scale structure.
4. Simplicity: A tool based on simple technology is easier to implement and adapt. This also increases chances that the tool is made available in commonly used software which in turn increases its spread.
5. Robustness: A tools dynamics should be able to handle drastic changes in the design gracefully. This enables the user to explore very different designs quickly.

While each of the properties enumerated above is important, most of them are in conflict with each other and therefore yield a trade-off. To develop a practical digital prototyping tool, one has to carefully evaluate the requirements of the targeted users.

### 1.2.3 Constrained Modeling

Modeling an architectural structure can be formalized as an instance of a constrained modeling task with specific types of constraints. At the core of most computer assisted constrained modeling is a combination of *Constrained Numerical Optimization* and a numerical representation of the geometry together with a formalization of the constraints of interest. A clever combination of those ingredients can make all the difference in terms of the properties enumerated in Section 1.2.2. In the following we first discuss the two fundamental parts of computational modeling, then the common types of constraints. We then describe three types of approaches to constrained modeling relevant in our context: rationalization, constraint-aware modeling and design space exploration. We list examples of fabrication constraints in Section 1.3.

### 1.2.4 Combinatorial and Continuous Modeling

Computational approaches to the modeling task can be divided into two conceptually different parts: *combinatorial* and *continuous* modeling. In the first step, the combinatorial modeling, also referred to as *discretization* or *remeshing* in the context of meshes, the parameter of a model and their interpretation have to be chosen. An example would be the connectivity of a triangle mesh. On a polygonal mesh, one would have to additionally define an interpolation scheme for the surface defined by non-planar polygons. Another commonly used discretization is a tensor-product surface including its control points. In the second step, the continuous modeling, an assignment of a concrete

value for each parameter has to be found such that all constraints are fulfilled. The combinatorial choice in the first step often has a big influence what can be achieved in the second step. Computational approaches to step one usually fall into the class of *Combinatorial Optimization* which are more difficult than their continuous counterparts because finding an optimum involves exhaustive searching in very large sets. Both steps could be combined into a single problem of mixed discrete-continuous nature.

### 1.2.5 Hard and Soft Constraints

The concept of constraints is often used in two fundamentally different meanings: *Hard* constraints express a *must-have*. The slightest deviation from a hard constraint renders the result meaningless. A typical example of a hard constraint is the static equilibrium of an architectural structure. *Soft* constraints express a *nice-to-have* and are also called *objectives*. They are often formalized as an objective function to be optimized which assigns a niceness measure to each result. Typical examples of soft constraints are smoothness or cost. Soft constraints can be converted into hard constraints by setting a hard limit on the objective function, for example, to formalize the fact that a flat glass panel can be slightly deformed without breaking. The importance of individual soft constraints can be weighted relative to each other. In contrast, each hard constraint needs to be satisfied completely. When formalizing a problem with hard constraints one has to be very careful to ensure that a solution exists at all. This however is often a hard problem in itself due to global coupling and non-linearity of constraints, and renders corresponding methods less popular.

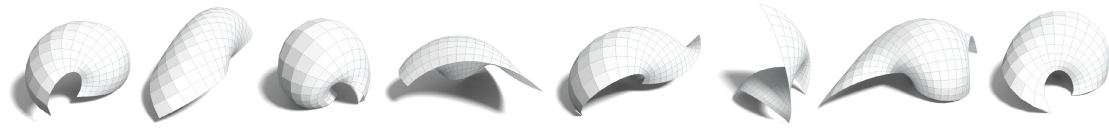
### 1.2.6 Rationalization and Paneling

*Rationalization* refers to the process of finding an approximation of a given freeform model which can be fabricated and built at reasonable cost. A rationalization usually follows as a post-processing step of an unconstrained freeform modeling session, often stressed by using the term *post-rationalization*. This approach is preferred by many designers who argue that the constraints limit their creativity. However, the approximation resulting from rationalization can deviate considerably and uncontrollably from the unconstrained input model. In contrast, a tool that directly integrates fabrication-awareness lets the designer explore and choose a final model in a more informed manner. *Paneling* is an instance of rationalization and refers to the approximation of a surface by a set of surface components, so-called panels, producible at reasonable cost. We propose a cost-optimizing paneling method in Chapter 4.

### 1.2.7 Constraint-aware Modeling

In contrast to rationalization, *constraint-aware modeling* refers to modeling approaches which directly integrate constraints, leading to a “what you see is what you get” experience for the user. Relevant to this thesis are more specifically fabrication-aware and statics-aware modeling. Note that a rationalization method fast enough to compute results at interactive rates would also yield a constraint-aware modeling tool. We present a constrained continuous fabrication-aware modeling framework in Chapter 3.

### 1.2.8 Design Spaces



**Figure 1.2** – *Various designs of a design space defined by a hard, co-circular constraint for each grid line, and soft squareness objective per quad. Figure from [BDS<sup>+</sup> 12].*

Design spaces provide an alternative view on constrained modeling. A *design space* contains all the sets of parameters of a numerical representation of a model (e.g. vertex positions of a mesh) that satisfy all hard constraints. Modeling under constraints can then be interpreted as an exploration of the corresponding design space. Soft constraints define an objective function for each design. The minima of this objective function in the design space correspond to the best designs among the constrained designs, as according to the soft constraints. Ultimately the user will have to choose a unique point out of the design space for an actual realization. As soon as a model exposes more than a few parameters, its design space gets high-dimensional, and therefore, very challenging to grasp and hard-to-impossible to keep a global overview or intuition. This is equally true for humans as for algorithms in general. Only in-depth geometric study could reveal global insights of a specific constraint set. The user can keep exploring locally, e.g. with explicit handles or low-parametric approximations, until he is satisfied with a solution. He can also add soft constraints to guide an algorithm towards the minimum, or he can add hard constraints to refine the design space. The latter however can quickly lead to contradicting constraints where the design space is empty.

## 1.3 Fabrication Constraints

One of the key benefits of building architectural structures by components (compared to monolithic ones) is fabrication: Each component can be produced in the most appropriate manner. The combination of material, fabrication-process and budget constrain each component and in turn the whole structure. Freeform concrete shells might appear as counter examples because they are a single component of material. The formwork

necessary for casting the concrete however often consists of components, for example the ground floor of the Rolex Learning Center at EPF Lausanne. In the following we discuss a few examples of fabrication-processes and their constraints.

#### 1.3.1 Cutting

Raw material can be cut apart by tools such as a knife, saw, water-jet or laser. If mounted onto a machine, a computer can directly control the tool, enabling precise reproduction of digital cuts and saving manual labor. A machine to cut flat pieces is particularly simple and many raw materials come in flat sheets. In this case, the cut component is constrained to be planar. Other machines can move the cutting tool or equivalently the raw material: A hot-wire cutter for example spans a hot piece of wire, which can melt through polystyrene. Assuming the tool can only cut straight lines, such a machine can be used to cut a particular type of surface, called a ruled surface. Cutting is often also necessary to bring raw material into the appropriate shape for other fabrication steps.

#### 1.3.2 Milling

Similar to cutting, a milling machine uses a tool, e.g. a spindle, to remove material from a raw piece. The material is milled away at the location of the tool. The subtractive nature of milling poses a first constraint on the result. Milling machines vary tremendously in size and their degrees of freedom in positioning and orienting their tool, which further limits producible pieces. Also, there is an inherent trade-off between the smoothness of the result, and the milling time and implied cost through the size of the tool in use.

#### 1.3.3 Bending and Folding

Bending is a continuous deformation that stretches the material only slightly. Cylindrical pieces for example can be produced with a roll parallel to a raw sheet of material. More generally, roll forming refers to the fabrication process resulting from concatenating rolls of various shapes and orientation. Folding is a particular way of bending, where curvature is concentrated into curvilinear folds. The company RoboFold develops industrial robots for folding curved sheet metal. Bending constrains the result to be a continuous, slightly stretching, so-called quasi-isometric deformation of the input. Often bending constraints are idealized to be non-stretching purely isometric deformations, because numerical and fabrication tolerances will satisfy them only approximately anyway.

### 1.3.4 Casting and Molds

In casting, a liquid material is cast into a mold where it solidifies. Constraints arise due to the separability of the solid result from the mold. Molds consisting of multiple parts can be carefully designed to ensure separability. A variant of casting is drawing, where a piece of flat material, usually metal, is drawn by a mold or robot in its solid state. This leads to constraints on the thickness of stretched material and overhang relative to the drawing direction. Other fabrication processes can be used to produce molds for casting. Reusable molds can be used to mass-produce many copies of a single piece.

### 1.3.5 3D Printing

3D printing is a very flexible fabrication process that has seen a huge development in recent years. 3D printers are used to print food, fully functional plastic and metal prototypes and even human tissue appropriate for surgery. Widespread interest has led to affordable consumer-level printers. Some 3D printers add material layer by layer, where each layer consists of a partially molten material similar to a 2D printer using thick ink. These types of additive 3D printers are limited in the overhang, which is often alleviated by adding support material or decomposing the target into multiple prints. The support material then has to be removed after printing. Other 3D printers proceed by binding part of a powder volume or respectively solidifying liquid. While 3D printers are very versatile, they tend to be slower and more expensive than other fabrication processes. 3D printers are often ideal for a small number of rapid prototypes, while casting would be used to mass-produce a design. The mold for casting could for example be 3D printed as well.

More fabrication-processes and geometric constraints are also discussed in our work on paneling freeform surfaces in Chapter 4, in particular in Table 4.1.

## 1.4 Assembly

In our context, assembly refers to the process of joining components together, resulting in an architectural structure. While individual components can usually be handled comfortably with common tools such as cranes and scaffolds, partially assembled substructures that arise during construction need to be held in place by temporary support. For traditional, box-like structures, it is usually quite obvious that a vertically increasing assembly order minimizes support. In a freeform structure however, finding a support-minimizing assembly order can be challenging. The temporary support, together with the common tools and labor of construction make assembly a major cost factor of a project.

The choice of temporary support, order and joints all depend on each other. For example, the joints used between neighboring components impose constraints on the stability of

partially assembled substructures and the temporary support. The joints also limit the directions in which components can be assembled, which in turn constrains the order.

### 1.4.1 Joints

Since joints are very practical and ubiquitous, they come in various flavors: slits, friction-based shapes such as LEGO<sup>®</sup>, holes and screws, glue, welding, mortar, nails and inelastic materials (e.g. ropes in a wooden raft). Some joints need or consist of additional material, e.g. glue or screws, while others are integral to the components to be joined, e.g. jigsaw puzzle pieces or the timber panels discussed in [Rob15]. The wooden joint types which can be unjoined easily also facilitate replacement of components which is useful for restoration of a structure. The blocks of historic masonry bridges are held in place purely by self-weight and friction. Intermediate substructures were often held in place by wooden support and ropes. We propose a method to compute a work-minimizing assembly sequence for masonry structures using chains instead of traditional formwork in Chapter 5.

Even if there are no physical joints present in self-supporting structures, the assembly order has to be carefully designed to ensure that each block can be slid in without colliding with the already assembled substructure. In practice, this can be alleviated by temporarily deforming the substructure slightly before sliding in the next block. In general, there is a trade-off between the necessary deformation and the restriction on assembly direction dictated by the joint.

In practice, architects and builders must take care of many more aspects of construction, some of which are the temporal synchronization with production, safety factors and external loads such as workers, wind or snow, collision-avoidance at the construction site, e.g. between the components lifted into place, the existing substructure and temporary support. A discussion of these aspects however would go beyond the scope of this thesis.

## 1.5 Computational Tools

Above, we discussed various aspects of fabrication-aware modeling and assembly of architectural structures. In particular, we list requirements for a digital prototyping tool, fabrication methods and their inherent constraints and discuss aspects of assembly including joints, support, and order. The ever-growing computational power available suggests that digital tools can alleviate the complexity of architectural structures burdening a designer. Given such a tool, the designer can focus on the main aspects of a structure, instead of staying busy with tedious and time-consuming tasks. The designer can explore feasible designs and may develop some intuition about the corresponding design space. Such a tool could also be used for rationalization.

A significant challenge in developing such a computational tool lies in finding the appropriate level of abstraction: The mathematical models of design, fabrication and assembly of architectural structures should cover a wide range of practical aspects for as many projects as possible. An abstract model would not cover many practical aspects, while a overly specialized model would only serve for a few projects.

### 1.6 Contributions

In this thesis we contribute various methods and frameworks towards a computational tool for composite structures.

1. A robust and extensible computational fabrication-aware modeling framework handling a multitude of constraints on subsets of points. In this framework architectural structures are modeled by any type of point-based geometry, and constraints such as planarity, strain-limits, angle-bounds and many more can act on various combinations of those points. Also the tool ShapeOp makes this framework available as an open-source project which can be used in other code, in particular in Rhino’s Grasshopper, a tool widely used by architects and designers for digital freeform modeling. The framework is presented in Chapter 3.
2. A method to enable mass-production of a freeform architectural surface by finding a set of molds and an assignment to a mold for each component. By defining the cost of each mold and component, the method can minimize the budget by deforming the input design slightly. The tolerable offset is defined by the user. The method was extended to handle sharp edges and to define tolerances locally, e.g. to stay closer to the original in highly visible regions, while allowing more deviation elsewhere. The method is presented in Chapter 4.
3. A method to find a work-minimizing assembly order for a given self-supporting structure. The temporary support is limited to a set of chains from each component to a few user-given anchor points. The amount of work is measured by the number of times a chain has to be added or removed. The method is presented in Chapter 5.

### 1.7 Publications

This thesis presents the content of three publications in detail:

Mario Deuss, Anders Holden Deleuran, Sofien Bouaziz, Bailin Deng, Daniel Piker, and Mark Pauly. Shapeop - a robust and extensible geometric modeling paradigm, 2015. [DDB<sup>+</sup>15]



Michael Eigensatz, Mario Deuss, Alexander Schiftner, Martin Kilian, Niloy J. Mitra, Helmut Pottmann, and Mark Pauly. Case studies in cost-optimized paneling of architectural freeform surfaces, 2010. [EDS<sup>+</sup>10]

Mario Deuss, Daniele Panozzo, Emily Whiting, Yang Liu, Philippe Block, Olga Sorkine-Hornung, and Mark Pauly. Assembling self-supporting structures, 2014. [DPW<sup>+</sup>14]

Also, the following publications were developed and written during the thesis, but are not presented in detail:

Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. Shape-up: Shaping discrete geometry with projections, 2012. [BDS<sup>+</sup>12]

Bailin Deng, Sofien Bouaziz, Mario Deuss, Juyong Zhang, Yuliy Schwartzburg, and Mark Pauly. Exploring local modifications for constrained meshes, 2013.[DBD<sup>+</sup>13]

Bailin Deng, Sofien Bouaziz, Mario Deuss, Alexandre Kaspar, Yuliy Schwartzburg, and Mark Pauly. Interactive design exploration for constrained meshes, 2015. [DBD<sup>+</sup>15]



## 2 Related Work

The work presented in this thesis concerns the modeling and assembly of freeform architectural structures. Our main goal is aiding designers to cope with the many constraints arising in fabrication-aware modeling while minimizing cost of fabrication and assembly with computational tools.

The paneling method presented in Chapter 4 and the fabrication-aware modeling framework in Chapter 3 both fall under the broader topic of constrained modeling. The former is a rationalization method enabling mass-production of various panel types which is typically applied only after unconstrained modeling, while the latter integrates various constraints into the modeling. We will first discuss related work in the context of constrained modeling, followed by work related to our assembly method proposed in Chapter 5.

A recent survey on existing solutions and open problems in architectural geometry [PEVW15] provides an in-depth discussion of many topics touched upon in this thesis.

### 2.1 Constrained Modeling

There exists a large body of research on constrained modeling. Here we only discuss a few that are applied to architectural structures. Methods for shape space exploration were introduced in [KMP07] and [YYPM11]. The latter has a particular focus on constrained meshes and was extended to support curve-based editing in [ZTY<sup>+</sup>13]. Recently, a more efficient method for exploration of meshes under various constraints was proposed in [TSG<sup>+</sup>14] and [JTT<sup>+</sup>15]. In many constrained modeling tools a local edit leads to a global change due to the coupling between constraints. This can be counter-intuitive and frustrating to a user. A computational method for local edits in the presence of constraints by means of linear bases in the design space is presented in [DBD<sup>+</sup>13]. Constrained space deformation, with the advantage of being independent of the underlying geometric representation, is considered in [SMB14]. Fabrication methods

and computational approaches considered in architectural geometry are also discussed in [PEVW15]. The term bidirectional modeling was coined in [Kil06]. It stresses that constraints are not only influenced by the design and fabrication, but also the other way around: the design can be informed by constraints—they can act as a driver for the design.

### 2.1.1 Combinatorial Modeling

Regular quadrilateral (quad) panels are an interesting alternative to triangular panels. The combinatorics of a panel layout have a big impact on qualities achievable in terms of regularity and constraints. The problem of combinatorial modeling with regular polygons is thoroughly discussed in the computer graphics community under the topic of surface and volume remeshing. Two recent surveys on quad-remeshing are available at [Pan15] and [BLP<sup>+</sup>13]. Hexagonal remeshing is addressed in [NPPZ12]. One important approach to remeshing involves computing  $n$ -symmetric tangential vector fields as discussed in [BZK09] and [KCPS13]. Volumetric remeshing with regular hexahedrons (cube-like elements) has been addressed in [LLX<sup>+</sup>12] and [KBLK14]. Constrained regular meshes are very restrictive for modeling, so modeling tools usually only consider the regularity as soft constraints. Alternatively hybrid meshes are considered, e.g. triangulated quad meshes or quadrangulated hexagons (see Figure 5 and 44 (right) in [PEVW15]).

Our paneling method is initialized from a polygonal mesh, but then uses a more flexible representation to allow gaps between panels and deviation from the input design.

### 2.1.2 Paneling

Paneling an architectural freeform surface refers to an approximation of a design surface by a set of panels. Approximations consisting of geometric primitives were considered in computer graphics as proxies for rendering or simulation and shape abstractions for analysis or modeling [CSAD04] and [YLW06]. These methods alternate between segmentation and approximation. In paneling, however, the seams between the panels need to be considered or designed explicitly because they are of high visual importance.

#### Planar Paneling

Most work on the paneling problem deals with planar panels. Based on the theory of discrete differential geometry (see also [BS08]), Pottmann and colleagues propose methods for covering general freeform surfaces with planar quad panels. Additionally, their panelings allow for simple, new ways of supporting beam layouts and multi-layer structures [LPW<sup>+</sup>06, PLW<sup>+</sup>07]. Their method was then extended to the covering of freeform surfaces by single-curved panels arranged along surface strips [PSB<sup>+</sup>08]. Figure

4.3 shows an example of a freeform surface rationalized using planar quads and developable strips, respectively. Combinatorial modeling, or remeshing, has also been addressed for planar quads [LXW<sup>+</sup>11] and hexagons [LLW15] and with further fabrication-constraints [JWWP14]. A generalization of  $n$ -symmetric tangential vector fields that can be used to compute conjugate fields leading to approximately planar panels once integrated was proposed in [DVPSH14].

### Developable Paneling

Other early contributions to the field of freeform architecture come from research at Gehry Technologies [She02]. These are mostly dedicated to developable or nearly developable surfaces, as a result of the specific design process that is based on digital reconstruction of models made from material that assumes (nearly) developable shapes. This approach is well suited for panels made of materials like sheet metal that may be deformed to developable or nearly developable shapes at reasonable cost. Panels made of materials like glass, however, limit affordable production processes to very restricted classes of developable surfaces (see Table 4.1).

### General Paneling

The approaches discussed above, however, focus on specific types of panels (planar or developable) for paneling a given freeform surface, and do not explicitly consider the aesthetic quality of panel layout or surface smoothness. With these rationalization approaches it is difficult to freely choose the paneling seams, since they need to closely follow a so-called conjugate curve network on the given freeform surface, which is defined by the curvature behavior of the surfaces (see [dC76] and [LPW<sup>+</sup>06]).

Our paneling method computes solutions by controlled deviation of the reference surface to increase the mold reuse for mass-fabrication. This is similar in spirit to symmetrization [MGP07, GPF09] proposed to enhance object symmetry, i.e., repetitions, by controlled deformation of the underlying meshing structure. Our method explicitly represents a curve-network to which the panels only need to fit approximately. Most approaches discussed above however use polygonal meshes to represent a paneling, which can overly constrain the problem. In fact, the parameters our method exposes to a user are thresholds on the closeness and angle between neighboring panels.

Another type of paneling uses so-called point-fold structures consisting of pyramidal panels. A method to maximize mold-reuse is discussed in [ZCBK12]. Extreme panelings consisting of repetitions of a single equilateral triangle are looked at in [HEB15].

### 2.1.3 Rationalization

Because freeform surfaces are visually exposed, high-quality panelings are very important. Rationalizations of other elements in architectural structures, however, have also been investigated: A rationalization method for structures consisting of circular arcs, congruent nodes and smooth panels have been proposed in [BPK<sup>+</sup>11]. Long-range components of architectural structures have been considered in the construction of timber rib shells, where straight wooden beams behave like geodesics when bent onto a given surface [WP06]. The work on functional webs discusses the layout of long-range components by looking at families of curves covering freeform surfaces under a variety of constraints [DPW11]. Approximations of structures using a limited set of primitives is studied on an example of the physical modeling tool Zometool in [ZLAK14].

### Parametric Modeling

A forward approach to rationalization is parametric modeling. An example for this was proposed by Glymph and coworkers [GSC<sup>+</sup>02], where certain classes of surfaces are rationalized using planar quadrilateral panels. Parametric modeling is also available in many standard computer-aided design tools nowadays. Such an approach introduces a logic into a geometric model by means of a generative sequence and relations between geometric objects. This logic helps in enabling simultaneous control of the surface shape and the paneling layout. The simple causal chains inherent to parametric modeling, however, are insufficient for the rationalization of complex freeform geometries.

### 2.1.4 Self-supporting Structures

Optimization of masonry structures is an active area of research in the computer graphics community. The shape of architectural models can be automatically adjusted to guarantee structural stability [WOD09, WSW<sup>+</sup>12]. Our assembly method uses the same model of statics for verifying structural stability, as formalized in [Liv92]. Much effort has been devoted to designing valid self-supporting shapes [VHWP12, LPS<sup>+</sup>13, dGAOD13, PBSH13], yet the issue of how to construct such structures from the ground up has been largely ignored. Cable elements were integrated in masonry design in [WSW<sup>+</sup>12] but with predetermined connectivity.

### 2.1.5 Constrained Numerical Optimization

For computational modeling, numerical optimization is a fundamental tool as it allows multiple requirements to be incorporated in the design process. For example, in architectural geometry, design shapes are often optimized according to certain geometric constraints that correspond to fabrication requirements [PEVW15]. In this section, we

discuss methods related to the solver we propose in our fabrication-aware modeling framework. Since a complete overview on constrained numerical optimization is out of the scope of this thesis, we refer the reader to [NW06] for further information on this subject.

### Non-linear Least Squares

Many constrained problems arising in architectural geometry are formulated as a non-linear least squares problems with each residual term corresponding to one constraint. This problem is then solved using standard solvers such as Gauss-Newton and Levenberg-Marquardt, or a penalty method based upon them, to obtain the final shape [PSB<sup>+</sup>08, SHWP09, ZSW10, PHD<sup>+</sup>10, DPW11, BPK<sup>+</sup>11]. Our paneling method uses this approach to solve the continuous subproblem in the alternating minimization.

### Force-based Solver

For form-finding, one popular approach is to model the shape as a system of nodes subject to internal and external forces, and to compute the final shape as an equilibrium state of the system [Day65, KO05, AAS<sup>+</sup>09, SP15]. For example, Kilian and Ochsendorf [KO05] use particle-spring systems for finding structural forms composing only axial forces. Using an implicit Runge-Kutta solver for computing the equilibrium state, their method allows the user to interact with the simulation while it is running. Such force-based approach is also adopted in Kangaroo, a live physics engine built on top of the computer-aided design tool Grasshopper [Pik13]. By modeling geometric constraints as forces, Kangaroo can perform not only form-finding and physics simulation, but also constraint solving and optimization, making it a popular tool among architects.

### Implicit Solver

Although force-based systems are intuitive to set up, it is challenging to simulate their behavior in an efficient, accurate, and stable way [WB97]. Implicit solvers allow for large time steps and require fewer iterations, but each iteration can be quite costly to compute since it requires solving a system of algebraic equations. Also, adding new forces requires the derivation of a Jacobian, making them more difficult to extend.

### Explicit Solver

Explicit solvers involve lower computational cost for each iteration, but at the same time require smaller step sizes to produce stable results, which can lead to a large number of iterations. For example, one issue of Kangaroo as presented in [Pik13] is that the simulation can explode for highly stiff problems, since such problems require a step

size much smaller than the default value; as a result, it is difficult to compute a shape that satisfies the given constraints exactly, because this will require large forces for the constraints and lead to very stiff systems.

## 2.2 Assembly

Assembly is a universal concept in production across disciplines. There exists a large variety of joining mechanisms, assembly order strategies and temporary support. In this section we limit our discussion to work related to computer graphics and freeform architectural structures. Interesting joints for timber construction leveraging the forces acting on the structure have been considered in a recent thesis [Rob15].

### 2.2.1 Masonry Building Fabrication

Historically, construction practices for masonry buildings involved elaborate timber-frame structures guiding the vaulted forms and further sub-structures for intermediate points of support [Fit61, Fal12]. In the construction of modern freeform shells, traditional methods are still in use, with wood panels cut according to section curves [Wen09]. Formwork can be reduced in thin tile vault construction [ROR<sup>+</sup>10, DRPB12], however, these methods rely on significant tensile strength of mortar in comparison to the light weight of the tiles. Our assembly method presented in Chapter 5 addresses the more general case of heavy masonry blocks where mortar strength must be neglected, and consequently, intermediate stages of construction require dense support structures.

While less common, in medieval vault structures, tensioned ropes were sometimes used to hold arch blocks in place [Fit61]. Tensioned elements have also been used as an alternative to formwork in the contemporary building industry. For example, the Arch-Lock system [Dre13] uses chains in the construction of Roman arch bridges, tunnels and vaults. This chain-based system was an inspiration for our assembly method, however our method greatly expands on the complexity and generality of chain supports, such that it can be applied to *freeform* shell construction.

### 2.2.2 Optimizing Construction Sequences

Constructability has been investigated in the context of 3D puzzles [LFL09, XLF<sup>+</sup>11, SFCO12], 3D assembly instructions [APH<sup>+</sup>03], and design with planar interlocking pieces [HBA12, SP13, CPMS14]. These methods address geometric constraints that ensure no piece is obstructed by the existing structure during assembly. Some aspects of stability have also been studied in these works, such as the rigidity of joints as a function of slit placement. In contrast, our assembly method tackles the constructability problem with a focus on equilibrium constraints and optimizations to simplify the assembly process.



### 2.2.3 3D Printing

Support structures are an essential component of the 3D printing process, needed to stabilize a model at all stages of fabrication [WWY<sup>+</sup>13]. While 3D printers use an additive method, building models layer by layer, our assembly method takes advantage of freedom in the construction sequence; blocks can be placed in arbitrary sequences, constrained only by connectivity. For 3D printing, several methods have been developed to determine areas of high stress in the final printed model [SVB<sup>+</sup>12, US13]. Strut and truss structures have been proposed as a solution to reducing internal stress and preventing breakage of the print material [SVB<sup>+</sup>12, WWY<sup>+</sup>13]. Our chain-based assembly method is intended for temporary support that can be easily added and removed.

A large body of work exists in optimizing for physical phenomena in fabrication-oriented design. For example, prescribed deformation behavior [BBO<sup>+</sup>10] and kinematic constraints [CTN<sup>+</sup>13] have been studied in the context of character design. A sparse set of strings is used in [STC<sup>+</sup>13] to animate actuated models. In our assembly method, we apply fabrication technology for creating physical prototypes, but our goal is rather directed at the assembly, while physical validity of the final shape is assumed at input. New printing technologies supporting large-scale 3D printing [HD14] could directly be applied to fabricate freeform blocks for masonry structures.

### 2.2.4 Rationalization and Assembly

Surface rationalization [SS10, FLHCO10, EKS<sup>+</sup>10] and decomposition in freeform sweeps [BPW14] can be used to create architectural tessellations that reduce construction costs. These approaches could potentially be used to optimize block tessellations, and thus benefit directly from our assembly method that can generate construction sequences for masonry structures composed of arbitrarily shaped blocks.



## 3 ShapeOp

### A Robust and Extensible Geometric Modeling Paradigm

We present ShapeOp, a robust and extensible geometric modeling paradigm. ShapeOp builds on top of the state-of-the-art physics solver [BML<sup>+</sup>14]. We discuss the main theoretical advantages of the underlying solver and how this influences our modeling paradigm. We provide an efficient open-source C++ implementation<sup>1</sup> together with scripting interfaces to enable ShapeOp in Rhino/Grasshopper and other tools. To evaluate the potential of ShapeOp we present various examples using our implementation and discuss potential implications on the design process.

#### 3.1 Introduction

Under the well established geometric modeling paradigms such as constructive solid geometry or spline-based modeling, polygonal mesh modeling yields a good tradeoff between expressibility - its many degrees of freedom allow to approximate an arbitrary design - and computational effort - its inherent linear interpolation reduces mathematical complexity. This has led to the development of various form-finding and modeling tools for the exploration of shape spaces of polygonal meshes. In our context we consider a shape space as a set of all designs that respect given geometric constraints dictated by aesthetic, fabrication and cost requirements. See Fig. for an example of a shape space.

Shape space exploration is typically facilitated by an optimization algorithm that negotiates a large number of complex and possibly conflicting constraints to satisfy the design goals. Numerical solvers for constraint satisfaction therefore play a fundamental role in shape exploration environments. A number of requirements on these solvers are essential for an effective design process, such as numerical robustness, computational efficiency, flexibility to handle a diverse set of design constraint, and extensibility to adapt to new

---

<sup>1</sup>[www.shapeop.org](http://www.shapeop.org)

design environments.

Existing shape exploration methods are often restricted by inherent limitations of their optimization approach. They might be tailored to a specific set of constraints, for example planarity of polygons, which can limit design flexibility. They exhibit numerical instabilities or slow convergence, which makes interactive modeling cumbersome. Last but not least, they are often closed, monolithic software, which makes adaptations or extensions in new design tasks difficult. We propose a new computational approach to geometric modeling and design that alleviates these limitations.

We adopt the physics solver proposed in [BML<sup>+</sup>14] that integrates a variety of constraints, dynamics and handle-based shape space exploration, and add projective constraints described in [BDS<sup>+</sup>12]. We refer to the combination as ShapeOp. In this chapter, we evaluate the potential of ShapeOp for design in a number of examples using Rhino’s Grasshopper as a graphical user interface. We also discuss and provide our implementation of ShapeOp, which effectively bridges the gap between computer graphics research and practical computational design, and acts as a open-source template for making research available. ShapeOp can also act as a building block for algorithms exploring further aspects of the shape space, e.g. adaptive meshing, evolutionary optimization and automatic constraint selection. The contribution of this chapter is three-fold:

1. We propose ShapeOp, a state-of-the-art unified and extensible constraint solver, and make it accessible to the architectural modeling community.
2. We describe and provide an efficient C++ open-source implementation of ShapeOp and an integration into Rhino’s Grasshopper using Python’s ctypes.
3. We highlight design applications and demonstrate the extensibility of ShapeOp in various examples.

In the remainder of this chapter, we first describe the main ingredients of the ShapeOp modeling approach and numerical solver. We then provide more details on the open-source implementation, show several design examples and discuss potential implications of our approach to design processes.

The two papers [BDS<sup>+</sup>12] and [BML<sup>+</sup>14] provide a thorough discussion about previous work related to ShapeOp. Other work related to ShapeOp is discussed in Section 2.1, in particular in Subsection 2.1.5.

### 3.2 Solver

ShapeOp is a physics engine as well as an optimization tool, designed for a set of points that are subject to physical and geometric constraints. In dynamic mode ShapeOp

simulates physics by preserving momentum. In static mode ShapeOp optimizes for an equilibrium solution, which converges considerably faster due to the absence of oscillations induced by momentum.

For constrained optimization, ShapeOp adopts the iterative solver of [BML<sup>+</sup>14], which models physical potentials as well as geometric constraints including the ones presented in [BDS<sup>+</sup>12] in a unified manner. Each iteration of the solver consists of a local step and a global step (see Fig. 3.1):

**Local Step** A candidate shape is computed for each set of points that are commonly influenced by a constraint. For a geometric constraint, this amounts to fitting to the points a shape that satisfies the constraint. For a physical constraint, this reduces to finding the closest point positions that have zero physical potential value. For example, in Fig. 3.1, each quad face of a mesh is subject to the constraint of being a square. Thus in the local step, a square is fitted to each face as its candidate shape.

**Global Step** The candidate shapes computed in the local step which might be incompatible. New point positions are computed, such that each set of points subject to a common constraint are as close as possible to the corresponding candidate shape (see Fig. 3.1 right).

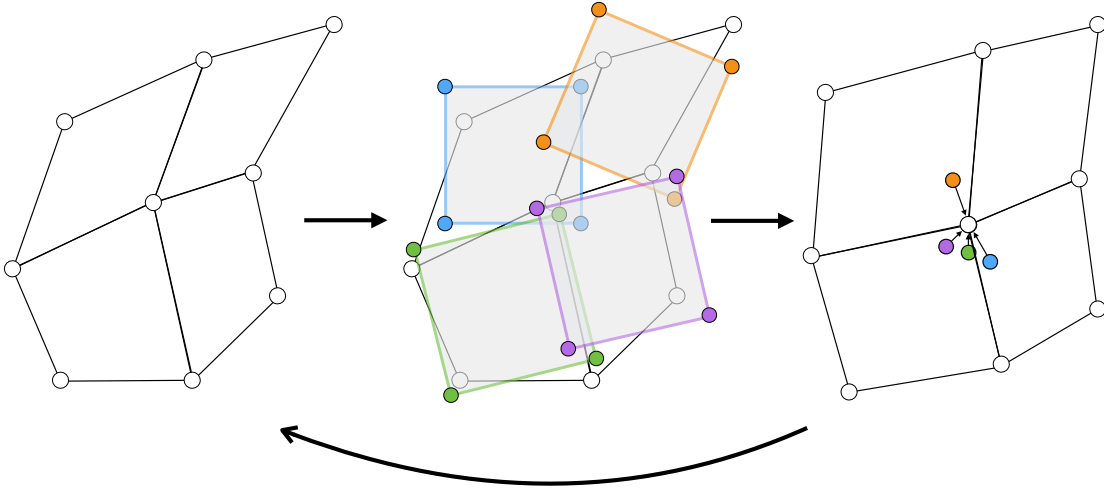
By repeating the above steps, the overall constraint violation is weakly decreased in each iteration, and the mesh converges to a shape that satisfies all physical and geometric constraints as much as possible. Moreover, each iteration can be run very efficiently: in the local step different constraints can be handled in parallel, while in the global step we only need to solve a linear system with a fixed matrix.

For simulating dynamics, ShapeOp uses the implicit Euler integration scheme from [BML<sup>+</sup>14], where at each integration step the physical and geometric constraints are resolved using the above local-global solver. Thanks to the efficiency of the local-global solver, ShapeOp benefits from the stability of implicit integration, with significantly lower computational cost than traditional implicit Euler solvers. ShapeOp also allows defining external forces such as wind and gravity.

For completeness of this chapter we include parts of the paper [BDS<sup>+</sup>12] in the following: Section 3.2.1 describes the general approach for constraint satisfaction based on projection. Section 3.2.2 adapts this approach to the domain of discrete geometry.

### 3.2.1 Proximity Function

We draw inspiration from a technique applied in the signal processing community for constraint satisfaction problems that may not have feasible solutions [Com94]. Central to



**Figure 3.1** – A quad mesh constrained to consist of squares illustrating the ShapeOp solver. Left: Initial configuration. Middle: Local step - Projecting each quad onto its closest square. Right: Global step - Joining the individual projections by a global minimization. The resulting mesh is then used as initial configuration and the solver iterates.

the method is a *proximity function* that measures the weighted sum of squared distances of a point to a collection of constraint sets, i.e. the sets containing feasible solutions to their respective constraints. For a collection of constraint sets  $\{C_1, C_2, \dots, C_m\}$ , let  $d_i(\mathbf{x})$  measure the 'least amount of change' in  $\mathbf{x} \in \mathbb{R}^n$  in order to satisfy the constraint  $C_i$ . The proximity function is then defined as

$$\phi(\mathbf{x}) = \sum_{i=1}^m w_i d_i(\mathbf{x})^2, \quad (3.1)$$

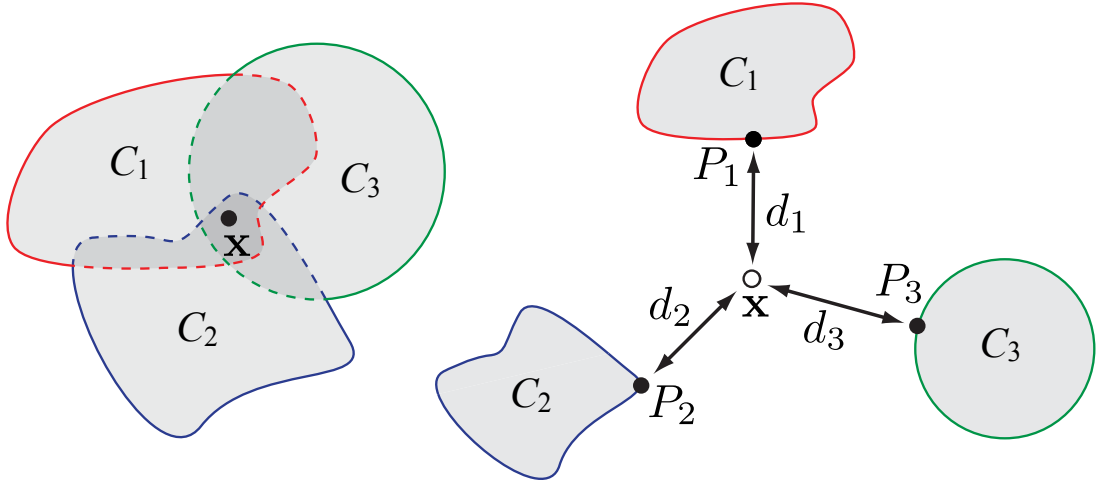
where  $w_i$  are non-negative weights that control the relative importance of the different constraints. Formally,  $d_i$  is the distance between a point  $\mathbf{x}$  and its projection  $P_i(\mathbf{x})$  onto the constraint set  $C_i$  (see Figure 3.2). We can formulate this projection as

$$\mathbf{y} = P_i(\mathbf{x}) = \operatorname{argmin}_{\mathbf{y} \in C_i} \|\mathbf{y} - \mathbf{x}\|_2^2, \quad (3.2)$$

which can be seen as moving  $\mathbf{x}$  in the minimal way to satisfy the constraint. The proximity function can now be written as

$$\phi(\mathbf{x}) = \sum_{i=1}^m w_i \|\mathbf{x} - P_i(\mathbf{x})\|_2^2. \quad (3.3)$$

This function encodes how well the constraints are satisfied through a distance measure. Finding a solution that minimizes the proximity function will therefore satisfy all the constraints if  $\phi(\mathbf{x}) = 0$ . Otherwise, a least-squares solution is obtained.



**Figure 3.2** – The proximity function  $\phi(\mathbf{x})$  is the weighted sum of squared distances  $d_i(\mathbf{x})$  of the point  $\mathbf{x}$  to the projections  $P_i(\mathbf{x})$  onto the respective constraint sets  $C_i$ . Minimizing  $\phi(\mathbf{x})$  yields a feasible solution if the constraint sets intersect (left), and a least-squares solution otherwise (right).

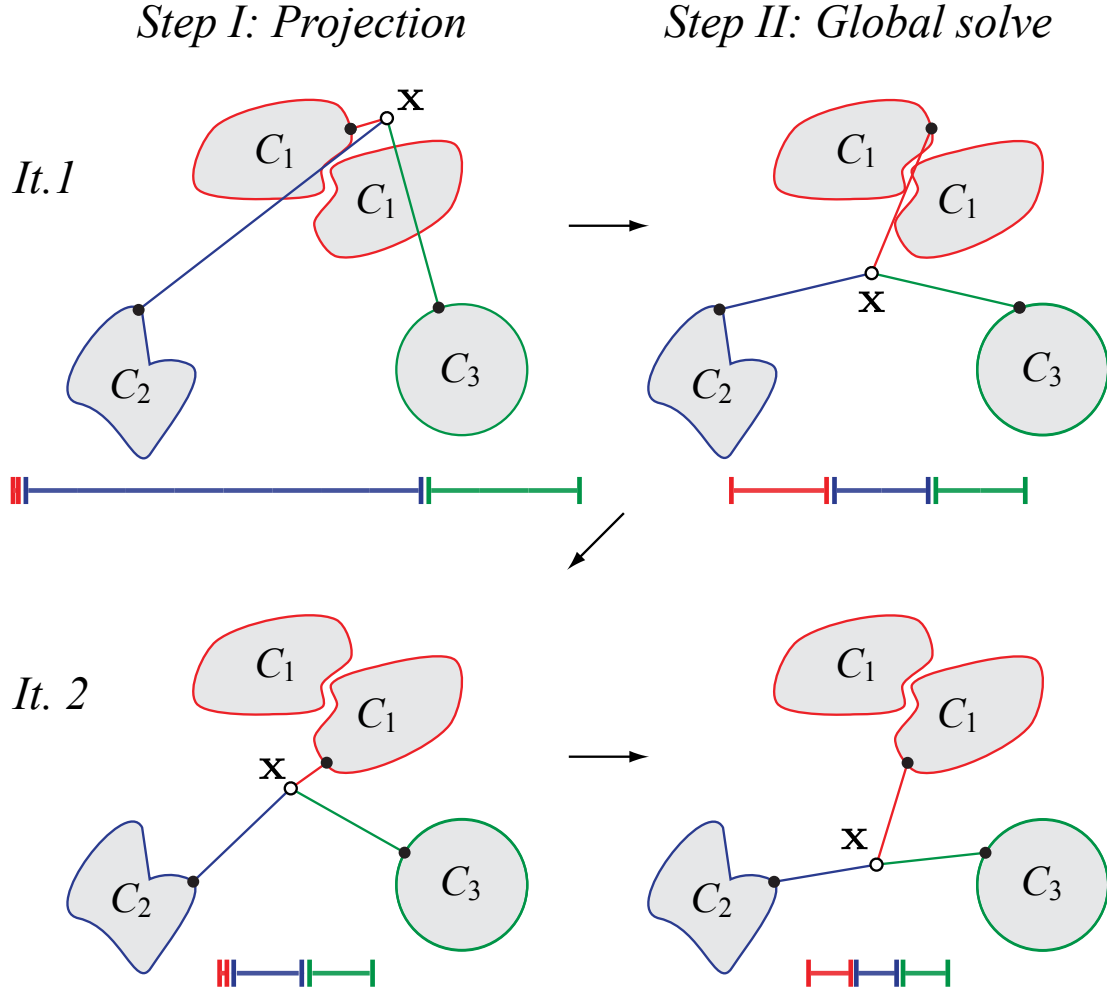
For linear projections  $P_i$  the global optimum is found using standard linear least-squares. Often, however, the projections are nonlinear and do not have an intuitive gradient. We therefore employ an iterative two-step minimization strategy:

- I Compute the projections  $P_i(\mathbf{x})$  using the current estimate  $\mathbf{x}$  (local step).
- II Update  $\mathbf{x}$  by minimizing Equation 3.3, keeping  $P_i(\mathbf{x})$  fixed (global step).

This scheme is guaranteed to converge monotonically to a local minimum, even though this minimum is not necessarily reached in a finite number of steps. The convergence rate depends on the conditions of the problem and the projection functions involved. To understand why the optimization converges, we observe that step I weakly decreases each constraint cost  $\|\mathbf{x} - P_i(\mathbf{x})\|_2^2$  given the current estimate  $\mathbf{x}$ , hence  $\phi(\mathbf{x})$  cannot increase. Step II minimizes Equation 3.3 globally for a fixed  $P_i(\mathbf{x})$ , thus  $\phi(\mathbf{x})$  also cannot increase. As a consequence, we obtain a sequence that is non-increasing and bounded from below (as mean-square errors cannot be negative), a sufficient condition for convergence to a local minimum. This argumentation is similar in spirit to the convergence proof exposed in [BM92] for the Iterative Closest Point (ICP) algorithm. The two step process is illustrated in Figure 3.3.

### 3.2.2 Shape Proximity for Geometric Data

Our key observation is that the proximity function is ideally suited to encode geometric shape constraints. The projection of a set of vertices onto a geometric shape is found by minimizing the sum of the squared distances of the vertices to the corresponding



**Figure 3.3** – Two iterations of the two-step minimization of the proximity function  $\phi(\mathbf{x})$  with  $w_i = 1$ . Step I computes the projections using the current estimate  $\mathbf{x}$ . Step II updates  $\mathbf{x}$  by minimizing  $\phi(\mathbf{x})$  keeping the projections fixed. At each step,  $\phi(\mathbf{x})$ , illustrated by the sum of the error bars, will decrease, even if some of the individual elements increase.

constraint set. This minimum is computed through shape matching, i.e. by finding the least-squares fit of the constraint shape onto the set of vertices. Let  $\mathbf{V}$  be a vector that stacks all vertices  $\mathbf{v}_1, \dots, \mathbf{v}_n \in \mathbb{R}^d$  of our  $d$ -dimensional data set and let  $\mathbf{V}_i \subseteq \mathbf{V}$  be the  $n_i$  vertices involved in shape constraint  $C_i$ . We formulate the shape proximity function as

$$\phi(\mathbf{V}) = \sum_{i=1}^m w_i \|\mathbf{N}_i \mathbf{V}_i - P_i(\mathbf{N}_i \mathbf{V}_i)\|_2^2, \quad (3.4)$$

where  $w_i$  are weights and  $P_i(\cdot)$  is the projection onto the constraint  $C_i$ , i.e. the corresponding least-squares fitted shape. The matrix  $\mathbf{N}_i$  is used to center the vertices of  $\mathbf{V}_i$



at their mean and is defined as

$$\mathbf{N}_i = (\mathbf{I}_{n_i \times n_i} - \frac{1}{n_i} \mathbf{1}_{n_i \times n_i}) \otimes \mathbf{I}_{d \times d}, \quad (3.5)$$

where  $\otimes$  is the Kronecker product and  $\mathbf{1}_{n_i \times n_i}$  is a  $n_i \times n_i$  matrix of ones. Subtracting the mean allows translational motion as a degree of freedom during the optimization. This introduces a global solve, but considerably improves convergence (see also Figure 10 of [BDS<sup>+</sup>12]). This formulation is possible because shape projections are invariant under translation. Equation 3.4 can be reformulated by rewriting  $\phi(\mathbf{V})$  as

$$E_{\text{shape}} = \phi(\mathbf{V}) = \|\mathbf{QV} - \mathbf{p}\|_2^2, \quad (3.6)$$

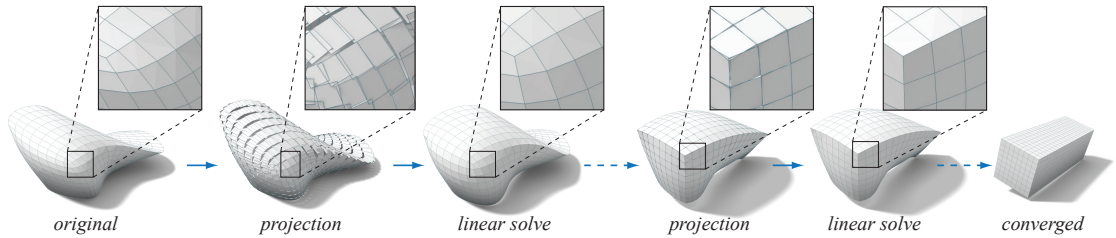
where the matrix  $\mathbf{Q}$  combines all weighted mean-centered constraint vertices, and  $\mathbf{p}$  integrates all projections. The alternating optimization scheme for each iteration then becomes:

- I For fixed  $\mathbf{V}$ , compute the projection vector  $\mathbf{p}$  using shape matching.
- II For fixed  $\mathbf{p}$ , solve the normal equations  $\mathbf{Q}^T \mathbf{QV} = \mathbf{Q}^T \mathbf{p}$  to update  $\mathbf{V}$ .

Since  $\mathbf{Q}$  only depends on the shape constraints, we can pre-factor the matrix  $\mathbf{Q}^T \mathbf{Q}$  using sparse Cholesky factorization. Figure 3.4 illustrates our two-step optimization scheme. In the projection step, we first compute the best fitting shape for each shape constraint. From the fitted shapes, we obtain the projected vertex positions and solve the linear system by back substitution using the prefactored matrix.

### 3.3 Projections

Central to the constrained optimization solver in ShapeOp are the so-called projection operators, which are used to compute the candidate shapes in the local step. Specifically,



**Figure 3.4** – Our optimization alternates between projection and linear solve. In this example, we prescribe a regular polygon constraint that pushes all quadrilaterals to become squares. The projection finds the best matching square for each quadrilateral to determine the target position for each vertex. The linear solve reconciles these projected positions in a least-squares sense.

given a set of points that are subject to a constraint, the projection operator finds the closest point positions that satisfy the constraint. A new constraint can be added easily to ShapeOp, as long as its projection operator is provided. No changes to global step of the solver are necessary to add a constraint.

### 3.3.1 Closeness

A simple example of a constraint is the closeness constraint: It is satisfied if the constrained vertex  $v$  coincides with a prescribed position  $c$ . Since the only way to satisfy the closeness constraint is by setting  $v$  equal to  $c$ , the projection  $P(\cdot)$  simply is given by  $P(v) = c$ .

### 3.3.2 Orientation

An example of a slightly more involved constraint is the orientation constraint: An orientation constraint acts on a set of vertices and is satisfied if those vertices all lie on a plane with a prescribed orientation. The orientation of a plane can be defined by a normal  $n$ . The constraint projection first subtracts the mean of the input set from each vertex. After doing so the least-square fitting plane with normal  $n$  contains the origin. The projection of a given vertex  $v$  onto the plane with normal  $n$  is then given by  $P(v) = v - n(n \cdot v)$ , where  $n \cdot v$  denotes the dot-product of  $n$  and  $v$ . A step-by-step tutorial on how to implement the orientation constraint in the source code and grasshopper component is provided in the ShapeOp documentation. See Table 3.1 for the constraints implemented in ShapeOp.

In the following we include the description of all projections presented in [BDS<sup>+</sup>12]. To simplify notation, we now denote with  $\mathbf{V} = \{\mathbf{v}_1, \dots, \mathbf{v}_n\}$  the vertices of a single constraint  $C_i$  (and not the full dataset) in the current configuration, and assume that these vertices are already mean centered. The original vertex positions are denoted by an apostrophe, i.e.  $\mathbf{V}' = \{\mathbf{v}'_1, \dots, \mathbf{v}'_n\}$ , and the projected vertex positions by a star, i.e.  $\mathbf{V}^* = \{\mathbf{v}^*_1, \dots, \mathbf{v}^*_n\}$ .

We describe three classes of constraints. *Continuous shapes*, such as planes or circles, *polygonal shapes*, such as line segments, regular polygons, or rectangles, and *relative shapes*. The latter encode the class of transformations that the shapes of the original geometry, e.g. polygons, tetrahedra, one-ring neighborhoods, etc., can undergo during the optimization. This allows the preservation of geometric properties such as lengths or angles of the original model.

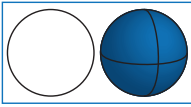
### 3.3.3 Continuous Shapes



#### Line - Plane

This constraint specifies that the vertices of  $\mathbf{V}$  should all lie on a continuous line or plane.

*Projection:* We can efficiently solve for the projection by first computing the sorted eigenvectors  $\mathbf{U} = [\mathbf{e}_1, \mathbf{e}_2, \mathbf{e}_3]$  of the  $3 \times 3$  covariance matrix  $\mathbf{C}^T \mathbf{C}$  where  $\mathbf{C} = [\mathbf{v}_1, \dots, \mathbf{v}_n]$ . We remove the last column of  $\mathbf{U}$  for plane projection and the last two columns for line projection. The projected vertices are then given as  $[\mathbf{v}_1^*, \dots, \mathbf{v}_n^*] = \mathbf{U}\mathbf{U}^T \mathbf{C}$ .

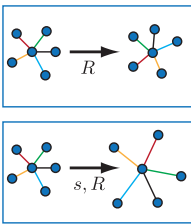


#### Circle - Sphere

This constraint specifies that the vertices of  $\mathbf{V}$  should all lie on a 2D circle or a 3D sphere.

*Projection:* Since the direct projection of 3D vertices to their 2D least-squares circle can be computationally expensive, we apply an approximate projection. We first project the vertices onto their least-squares plane (see above) and then fit a 2D circle within that plane. Circle fitting is achieved by minimizing  $\sum_j (||\mathbf{v}_j - \mathbf{c}||_2^2 - r^2)^2$ , where  $r$  and  $\mathbf{c}$  are the unknown radius and center of the circle, respectively. We solve for these parameters using the closed-form solution of [TC89] and project the vertices of  $\mathbf{V}$  onto this circle to obtain  $\mathbf{V}^*$ . The projection onto a sphere is computed by minimizing the same equation directly on the 3D points.

### 3.3.4 Relative Shapes



#### Rigid - Similar

These constraints are defined relative to the original vertex set  $\mathbf{V}'$ , i.e. they constrain the type of transformation that the vertex set can undergo. *Rigid* aims at restricting the deformations to isometries, while *Similar* aims for a conformal deformation.

*Projection:* Finding the closest rigid transform or similarity that maps the original vertices  $\mathbf{V}'$  onto the current set  $\mathbf{V}$  can be solved using the method described in [Ume91]. The algorithm computes the rigid transformation and uniform scale using least-squares fitting and allows a minimal and maximal scale constraint by keeping the rigid transformation as is and clamping the scale to the desired range.

While this approach works well, we also propose a faster projection operator for 2D shapes. The idea is to first project the vertices onto their least-squares plane and then

formulate the fitting in 2D. We denote the projected 2D points by a bar, e.g.  $\bar{\mathbf{v}}'_j$  is the projection of the original vertex  $\mathbf{v}'_j$  onto the least-squares plane.

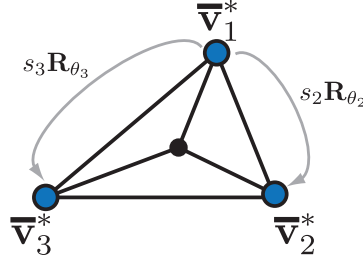
Let  $\mathbf{M}$  be all the sets of points conformal to the 2D points  $\bar{\mathbf{V}}' = \{\bar{\mathbf{v}}'_1, \dots, \bar{\mathbf{v}}'_n\}$ . We first find the point set  $\bar{\mathbf{V}}^* \in \mathbf{M}$  closest to  $\bar{\mathbf{V}}$ , i.e. solve for

$$\{\bar{\mathbf{v}}_1^*, \dots, \bar{\mathbf{v}}_n^*\} = \underset{\bar{\mathbf{V}}^* \in \mathbf{M}}{\operatorname{argmin}} \sum_{j=1}^n \|\bar{\mathbf{v}}_j^* - \bar{\mathbf{v}}_j\|_2^2. \quad (3.7)$$

As explained in [Hor87], at the minimum of Equation 3.7 the centroids of  $\bar{\mathbf{V}}$  and  $\bar{\mathbf{V}}^*$  coincide. Therefore, if  $\bar{\mathbf{V}}$  is centered, Equation 3.7 can be expressed as

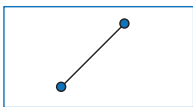
$$\underset{\bar{\mathbf{v}}_{1x}^*, \bar{\mathbf{v}}_{1y}^*}{\operatorname{argmin}} \left\| \underbrace{\begin{bmatrix} \mathbf{I}_{2 \times 2} \\ s_2 \mathbf{R}_{\theta_2} \\ \vdots \\ s_n \mathbf{R}_{\theta_n} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \bar{\mathbf{v}}_1^* \\ \bar{\mathbf{v}}_2^* \\ \vdots \\ \bar{\mathbf{v}}_n^* \end{bmatrix}}_{\mathbf{b}} \right\|_2^2, \quad (3.8)$$

where  $s_i \mathbf{R}_{\theta_i}$  represent the scale and rotation mapping the first point to the  $i$ th point in the original centered set  $\bar{\mathbf{V}}'$ .



The minimum  $\mathbf{x}$  of Equation 3.8 is obtained by solving the normal equation  $\mathbf{x} = (\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$ . We can then express the projection as a linear operator  $\mathbf{P} = \mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T$ , which maps the current point set  $\bar{\mathbf{V}}$  to the closest point set  $\bar{\mathbf{V}}^*$  in  $\mathbf{M}$ . The matrix  $\mathbf{P}$  depends only on the original point set  $\bar{\mathbf{V}}'$  and can thus be precomputed. If  $\mathbf{P}$  is applied to any point set in  $\mathbf{M}$ , by the idempotence property of the projection operator, the result is unchanged. Since  $\mathbf{A}^T \mathbf{A}$  is a  $2 \times 2$  matrix, this projection operator has a closed form expression.

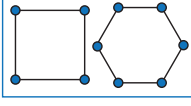
#### 3.3.5 Polygonal Shapes



##### Line Segment

For a pair of vertices  $\{\mathbf{v}_1, \mathbf{v}_2\}$ , this constraint specifies the allowed value for their relative distance.

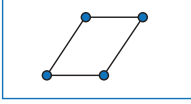
*Projection:* Let  $d = \|\mathbf{v}_1 - \mathbf{v}_2\|_2$  be the current distance between the vertices and  $d^*$  the desired length of the line segment. Then the projection  $\{\mathbf{v}_1^*, \mathbf{v}_2^*\}$  is computed as  $\mathbf{v}_1^* = \frac{d^*}{d} \mathbf{v}_1$  and  $\mathbf{v}_2^* = -\mathbf{v}_1^*$ .



#### Regular Polygon

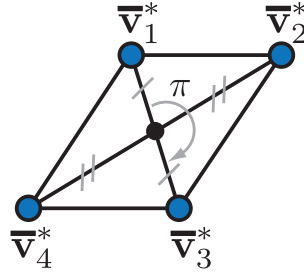
This constraint specifies that the vertex set  $\mathbf{V}$  should assume the shape of a regular polygon, i.e. have all angles be equal and all sides be of equal length.

*Projection:* Since a regular polygon is invariant only under similarity transformations, we can use the same projection method as described above for *relative shapes*. We simply replace the original vertex set  $\mathbf{V}'$  by the vertices of the regular polygon of the corresponding order.



#### Parallelogram

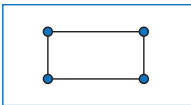
This constraint specifies that a quadrilateral should become a parallelogram, i.e. have two pairs of parallel sides.



*Projection:* We formulate the parallelogram fitting by extending the projection for *relative shapes* as described above. We first project the vertices onto their least-squares plane, then formulate the optimization as

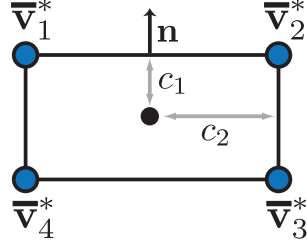
$$\operatorname{argmin}_{\mathbf{v}_1^*, \mathbf{v}_2^*} \left\| \underbrace{\begin{bmatrix} \mathbf{I}_{4 \times 4} \\ -\mathbf{I}_{4 \times 4} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} \mathbf{v}_1^* \\ \mathbf{v}_2^* \end{bmatrix}}_{\mathbf{x}} - \underbrace{\begin{bmatrix} \bar{\mathbf{v}}_1 \\ \bar{\mathbf{v}}_2 \\ \bar{\mathbf{v}}_3 \\ \bar{\mathbf{v}}_4 \end{bmatrix}}_{\mathbf{b}} \right\|_2^2. \quad (3.9)$$

As previously, the solution of this optimization is  $\mathbf{V}^* = \mathbf{A}(\mathbf{A}^T \mathbf{A})^{-1} \mathbf{A}^T \mathbf{b}$ .



#### Rectangle

This constraint specifies that a quadrilateral should become a rectangle, i.e. have only right angles.



*Projection:* We first project the vertices onto their least-squares plane and then fit the rectangle in 2D. Unlike the other polygonal shapes, we compute the equation of the four lines that define the rectangle by solving

$$\underset{c_1, c_2, \mathbf{n}}{\operatorname{argmin}} \left\| \underbrace{\begin{bmatrix} 1 & 0 & \bar{v}_{1x} & \bar{v}_{1y} \\ 1 & 0 & \bar{v}_{2x} & \bar{v}_{2y} \\ 0 & 1 & \bar{v}_{2y} & -\bar{v}_{2x} \\ 0 & 1 & \bar{v}_{3y} & -\bar{v}_{3x} \\ -1 & 0 & \bar{v}_{3x} & \bar{v}_{3y} \\ -1 & 0 & \bar{v}_{4x} & \bar{v}_{4y} \\ 0 & -1 & \bar{v}_{4y} & -\bar{v}_{4x} \\ 0 & -1 & \bar{v}_{1y} & -\bar{v}_{1x} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} c_1 \\ c_2 \\ \mathbf{n}_x \\ \mathbf{n}_y \end{bmatrix}}_{\mathbf{x}} \right\|_2^2 \quad \text{s.t.} \quad \|\mathbf{n}\|_2^2 = 1. \quad (3.10)$$

This optimization is minimized by taking the QR decomposition of  $\mathbf{A}$  and solving a  $2 \times 2$  eigenvalue problem as described in [GH95]. We then find the projected points by computing the intersection of these four lines.

While many of the constraints intuitively apply to specific primitives, some of them can be applied to an arbitrary set of points defining novel shape spaces. For example, the circle constraint was often applied to all quads of a mesh because of its desirable offset properties [PLW<sup>+</sup>07]. However, it can also be applied to each grid line of a quad mesh, defining an interesting shape space as illustrated in Figure 1.2. Similarly, the plane constraint is often applied to all polygons of a mesh to allow for fabrication by cutting planar material. But alternatively, one could apply a plane constraint to a point and all its immediate neighbors, yielding a smoothness constraint depending much less on the mesh than the laplacian constraint. Also note that the ShapeOp Solver has no explicit knowledge of a mesh, but only of a list of points. If an application or its user tries to constrain mesh primitives like triangles or quads, it has to provide ShapeOp with the indices for the list of points corresponding to the primitive when adding a constraint. This allows to apply ShapeOp to any set of points, e.g. mixing different geometric primitives such as splines, tetrahedral meshes, bezier patches or triangle soups, that are parametrized by point positions.

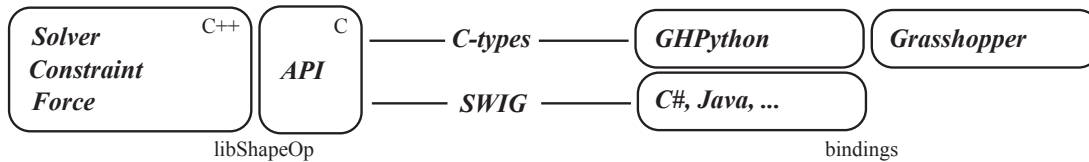
Constraint	Description
Edge Strain	Bounds the length of an edge ([BML <sup>+</sup> 14], § 5.1).
Triangle Strain	Bounds the strain of a triangle with respect to its initial configuration ([BML <sup>+</sup> 14], § 5.1).
Tetrahedron Strain	Bounds the strain of a tetrahedron with respect to its initial configuration ([BML <sup>+</sup> 14], § 5.1).
Area	Bounds the area of a triangle ([BML <sup>+</sup> 14], § 5.2).
Volume	Bounds the volume of a tetrahedron ([BML <sup>+</sup> 14], § 5.2).
Bending	Bounds the change in mean-curvature ([BML <sup>+</sup> 14], § 5.4).
Closeness	Constrains a point to a prescribed position (Section 3.3.1).
Line	Constrains points to a lie on a line (Section 3.3.3).
Plane	Constrains points to a lie on a plane (Section 3.3.3).
Circle	Constrains points to a lie on a circle (Section 3.3.3).
Sphere	Constrains points to a lie on a sphere (Section 3.3.3).
Rectangle	Constrains four points to form a rectangle (Section 3.3.5).
Parallelogram	Constrains four points to form a parallelogram (Section 3.3.5).
Uniform Laplacian	Constrains a point to the average of its neighbors ([BDS <sup>+</sup> 12], § 4).
Uniform Laplacian of Deformation	Constrains a deformation vector with respect to the initial position to be the average of its neighboring deformation vectors ([BDS <sup>+</sup> 12], § 4).
Rigid	This constraint is equivalent to Similarity, only that it does not allow for uniform scaling (Section 3.3.4).
Angle	Bounds the angle formed by three points ([DBD <sup>+</sup> 15], § 3.3.2).
Similarity	Constrains points to be similar to one of the prescribed set of points. Two sets of points are similar if there exists a rotation, translation and uniform scaling that maps one set onto the other. The similarity constraint automatically selects the closest of the prescribed sets of points to project to at each iteration (Section 3.3.4).

**Table 3.1** – *Constraints implemented in ShapeOp.*

The projections of [BDS<sup>+</sup>12] also been adapted in a more advanced and complex optimization enabling hard constraints in [DBD<sup>+</sup>15]. Their solver however is substantially more involved and would undermine the simplicity of ShapeOp.

### 3.4 Implementation

Our implementation of ShapeOp is distributed as a header-only C++ library. While it is possible to develop C++ plugins for computational design environments this requires a larger and substantially more involved development investment than what is offered by higher level programming languages provided in .NET compliant CAD environments such as Rhino 3D and Revit. Here languages such as C#, VB and Python make development of computational design models fast, responsive, and interchangeable. Below we demonstrate how we have integrated the ShapeOp C++ library directly into the Rhino/Grasshopper environment using Python scripting components. A selection of examples will demonstrate how Grasshopper users can start to implement ShapeOp in their definitions.



**Figure 3.5** – Schematic overview of our implementation of ShapeOp.

The implementation can be conceptually divided into two components: The core library `libShapeOp` with various bindings, and `applications` which use `libShapeOp`. The core library contains the abstract C++-classes `Solver`, `Constraint` and `Force`, and many classes deriving from and implementing them. The C-API provides an interface using C only, which simplifies calling `libShapeOp` from other code or programs considerably. ShapeOp also provides everything necessary to use SWIG<sup>2</sup>, a software development tool that can generate a multitude of bindings for `libShapeOp`. The applications contains the Grasshopper definitions using `libShapeOp`. The definitions use GhPython<sup>3</sup> to enable Python in scripting components. Inside the component we use Python’s `ctypes` to directly call `libShapeOp`.

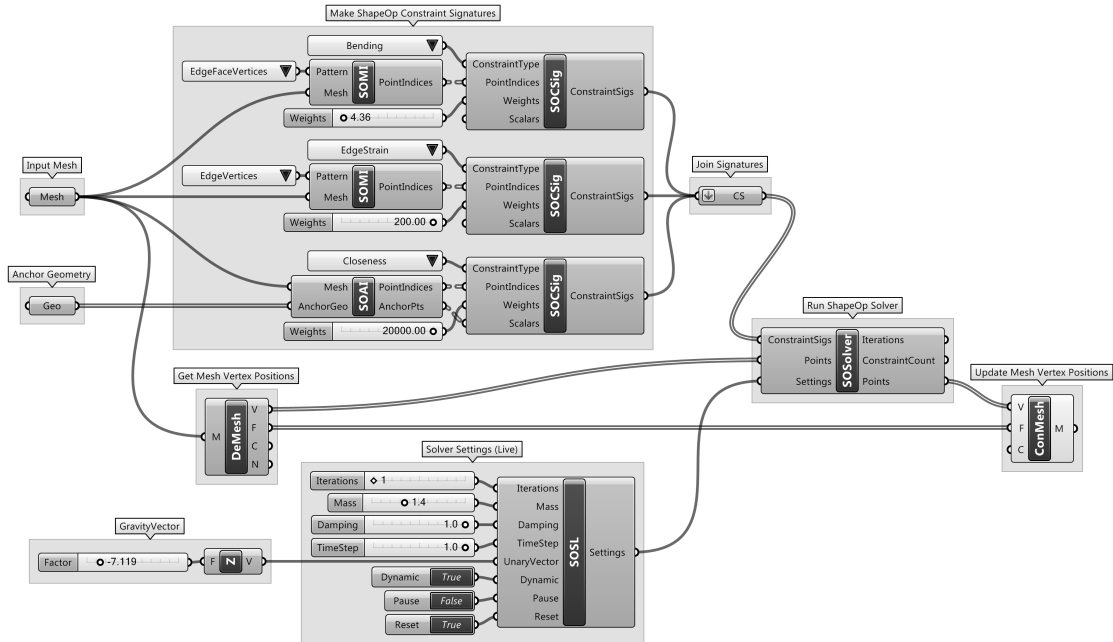
There are six ShapeOp Grasshopper components in the current release. The functionality of each is implemented in a Python script and can be viewed and edited by double-clicking a component. The `ShapeOp ConstraintSolver` (`S0Solver`) is the central component and is the only one that calls the `libShapeOp` library. It sends points and constraint signatures to the library and retrieves the result. The `ShapeOp SettingsLive` (`S0SL`) and `ShapeOp SettingsStatic` (`S0SS`) components are used to edit the ShapeOp solver settings and run the solve process. A constraint signature contains all the necessary

<sup>2</sup>[www.swig.org](http://www.swig.org)

<sup>3</sup>[www.food4rhino.com/project/ghpython](http://www.food4rhino.com/project/ghpython)



information to setup a constraint: The constraint type represented by a string; the indices of points to be constrained with respect to the global list of points; the weight of this constraint; the scalars, a list of floating point numbers encoding additional settings of the constraint.

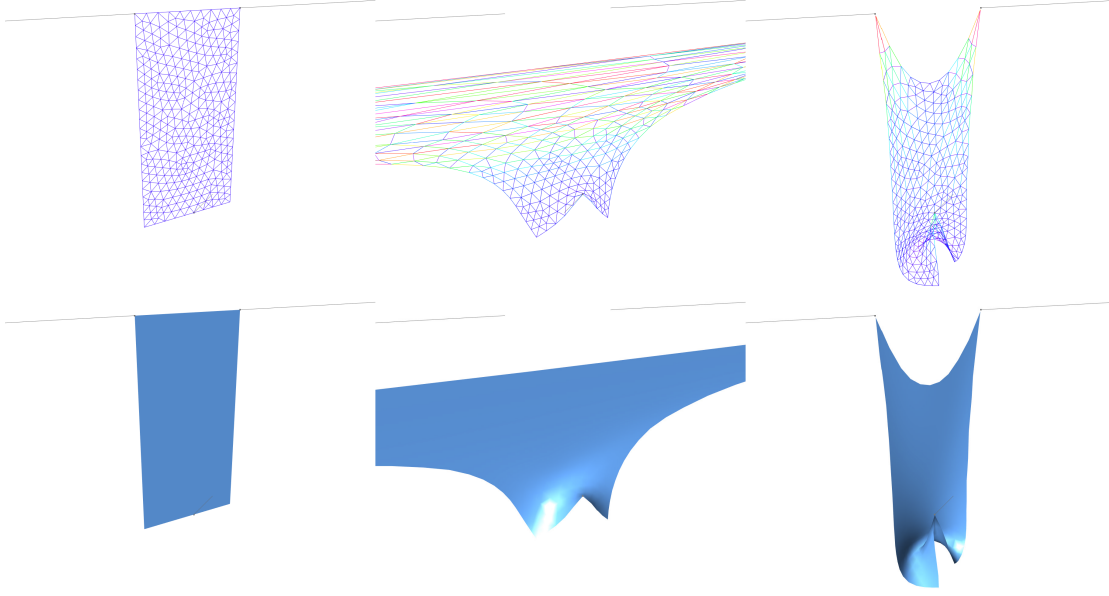


**Figure 3.6** – The Grasshopper definition used for the hanging cloth example seen in Fig. 3.7.

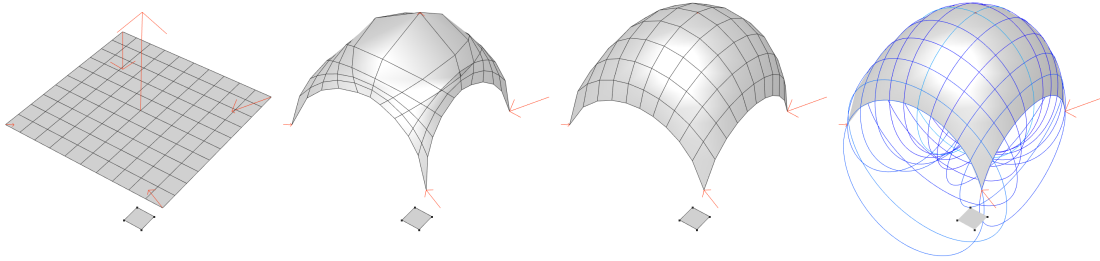
ShapeOp ConstraintSignature (SOCSig) constructs the constraint signatures. Constraint signatures are implemented using a Python dictionary, so they could also be created by custom python components other than SOCSig. ShapeOp MeshIndexer (SOMI) is a utility component to extract point indices from a mesh according to a provided pattern represented by a string. The resulting point indices are represented by a Grasshopper data-tree, which is equivalent to a list of lists, and could again come from any other script or Grasshopper component if desired. Note that SOCSig creates multiple constraints if the inputs are lists. For example, the two upper SOMI and SOCSig components produce a bending constraint with weight 4.36 for each edge in the mesh that is shared by two triangles. ShapeOp AnchorsIndexer (SOAI) is a utility component for setting up anchor constraints and translating them to closeness constraints. It picks the closest point in the mesh to the provided anchor point.

### 3.5 Examples

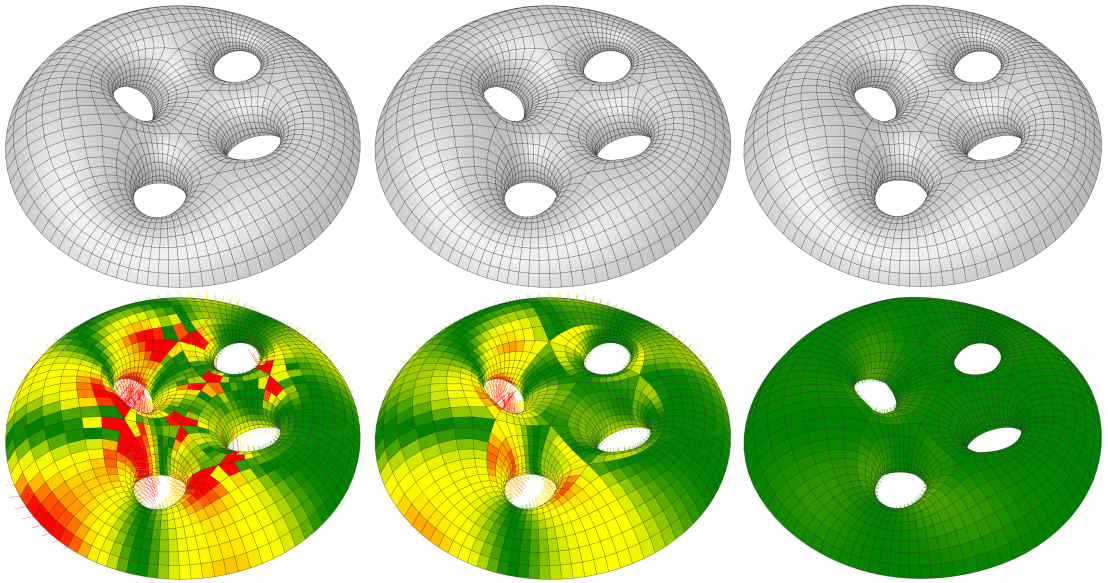
In Figs. 3.7 to 3.14, we provide some examples of using ShapeOp in Rhino/GhPython for different applications, including physics simulation, constrained modeling, rationalization, and form-finding.



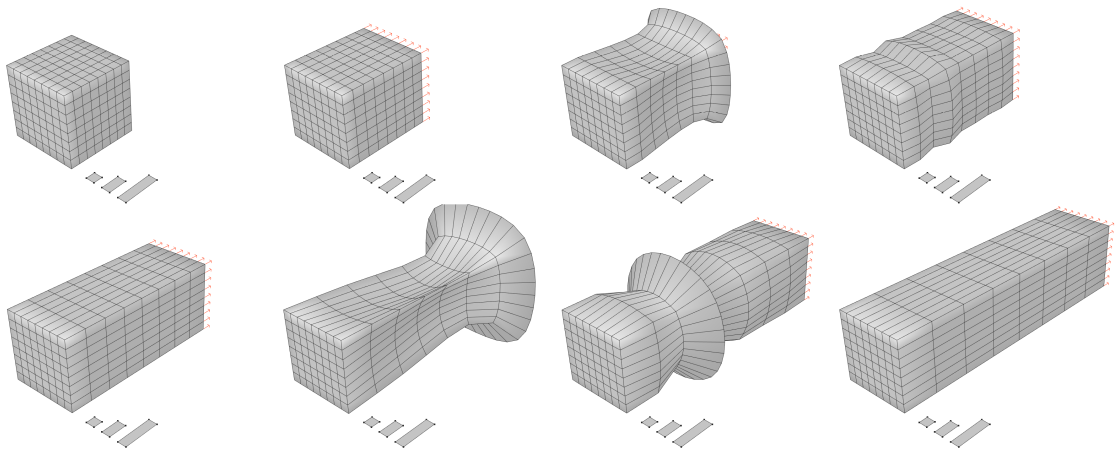
**Figure 3.7** – Use of ShapeOp for physics simulation of elastic materials. A hanging cloth modelled using edge strain and bending constraints. The three vertices are anchored using closeness constraints and all points are subjected to a gravity load. Left: The input mesh. Middle: The constrained mesh at the first solve iteration in which the anchors are immediately moved very far apart. Right: The constrained mesh after 100 iterations with the anchor point moved back to their starting positions. Top: Wireframe rendering with the edges coloured by their strain (red = high, blue = low). Bottom: Shaded rendering. The example demonstrates both the stability and the fast convergence of the solver.



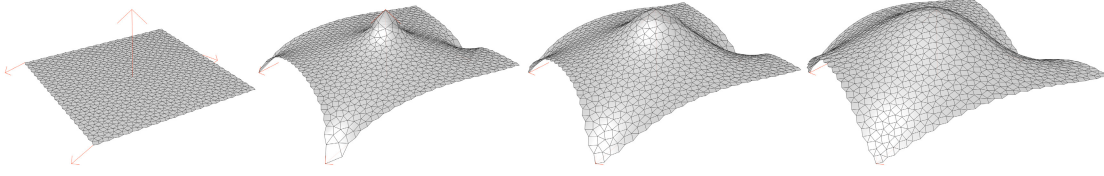
**Figure 3.8** – Use of ShapeOp for constrained modeling of a shell with rational geometric properties. The vertices on the parameter lines of a quad-mesh are constrained to always lie on a circular arc using the circle constraint. Each face is constrained towards being square using the similarity constraint. Five vertices are anchored to different positions than their initial positions, enabling shape exploration. Left to right: 1) The input mesh, the face used for similarity and vectors visualizing start/end positions for the anchors. 2) The constrained mesh after 10 iterations. 3) The constrained mesh after 300 iterations. 4) The constrained mesh with circles drawn through each of the parameter line vertices (Red = Line vertices distance to circle is large, Blue = Line vertices distance to circle is small).



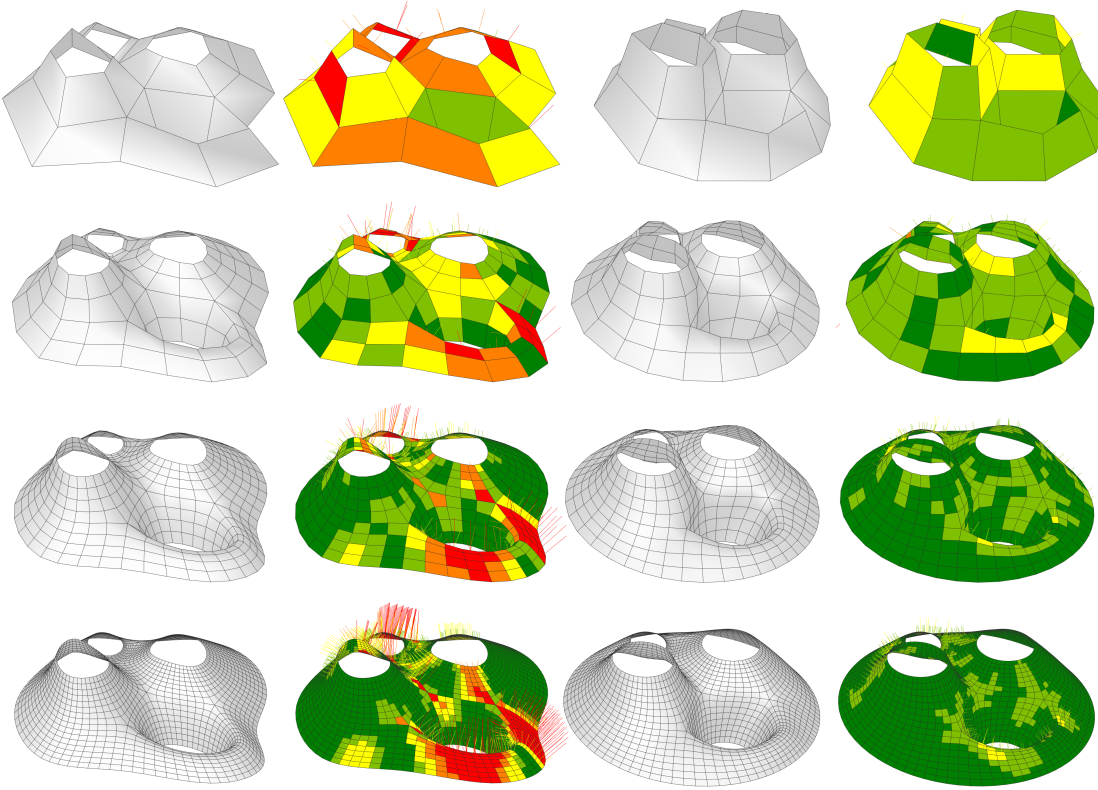
**Figure 3.9** – Use of ShapeOp for rationalizing an existing geometry. Each face of the quad-mesh is constrained towards being planar using the the plane constraint. Each vertex is constrained to its initial position using the closeness constraint by a small weight to maintain the shape of the mesh. Left: Input mesh. Middle: The constrained mesh after 10 iterations. Right: The constrained mesh after 200 iterations. Top: Shaded rendering. Bottom: Planarity analysis rendering (Red = Low planarity, Green = High planarity).



**Figure 3.10** – Use of ShapeOp for constrained modeling of box shape with multiple rigid shape targets. A quad-mesh box is anchored at the vertices on two sides of the box. The image sequence shows the vertices on one side being pulled away over time. As this occurs each mesh face attempts to project itself onto one of the three shape targets below the box. The solver has been initialized using dynamics leading to the rippling effect as the faces switch their projection targets from short to medium to long. This projection type is enabled using the rigid constraint.

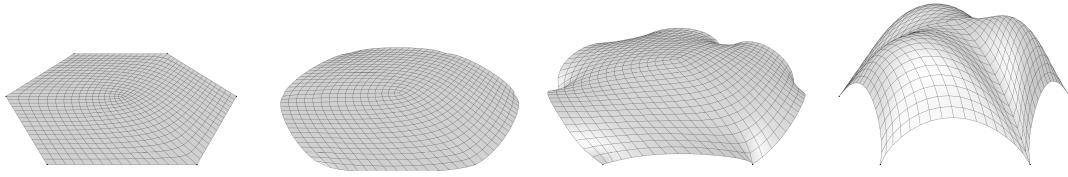


**Figure 3.11** – Use of ShapeOp for constrained modeling of a shell with topologically different shape targets. A planar mesh composed of both triangles and quads is anchored at four vertices using the closeness constraint. Using the similarity constraint, each face is constrained towards their initial shape i.e. an equilateral triangle or a square. 1) The input mesh. 2) The mesh after 1 iteration. 3) After 100 iterations. 4) After 500 iterations.

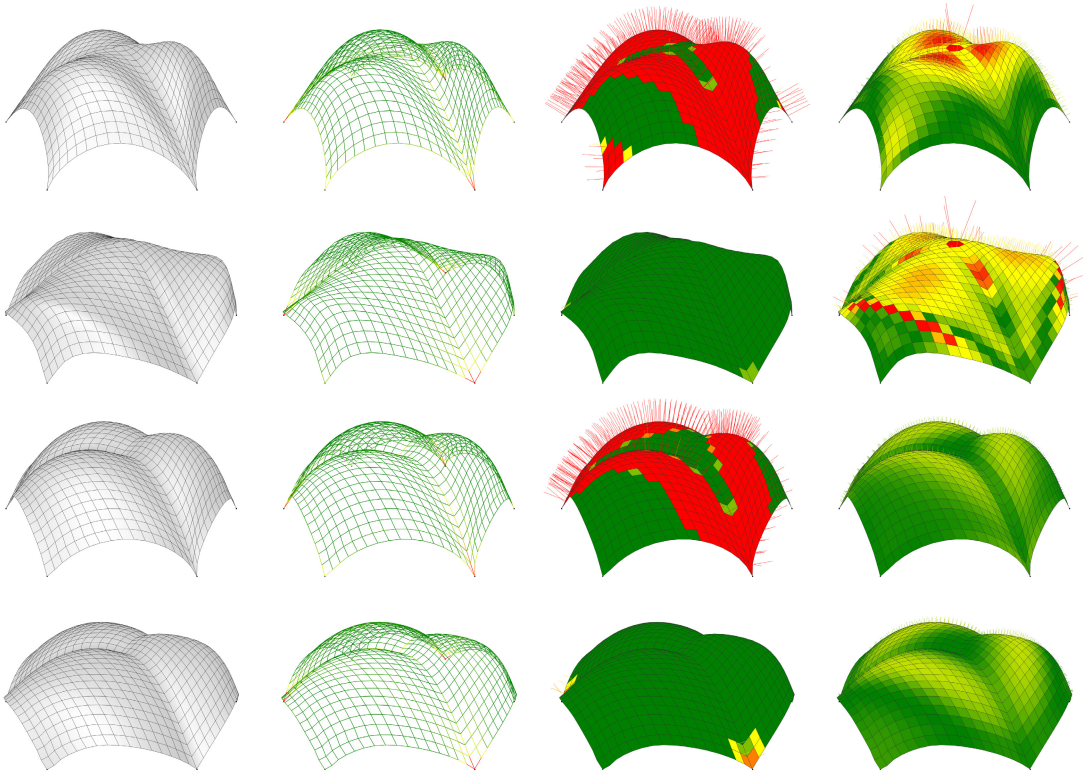


**Figure 3.12** – Use of ShapeOp for constrained modeling of a randomly generated quad-mesh with multiple constraints as design drivers. The example demonstrates the effect of applying the same constraints on meshes with different resolutions. It uses three primary constraints: 1) Limit the internal angles of each face to be within 80 and 110 degrees, 2) The boundaries of the mesh should lie on circles, 3) Each face should preserve its area. Additionally, a Laplacian of displacement constraint is added which smoothens out the mesh while maintaining the shape, and a bending constraint is added which ensures that face-face angles do not become too acute. The color code is a based on scoring system: The internal angles for each face are calculated. If an angle is within the desirable range it is scored 0, else 1. The scores are added for each face, best face score is 0 (Dark green) worst is 4 (Dark red).





**Figure 3.13** – Use of ShapeOp for funicular form finding. A hexagonal quad-mesh is anchored at each corner and subjected to an inverse gravity load. In this image sequence the only other constraint is that each edge should be 2.0 units long. This is implemented using the edge strain constraint. 1) The input mesh. 2) The constrained mesh after 1 iteration. 3) The constrained mesh after 10 iterations. 4) The constrained mesh after reaching equilibrium at iteration 1000.



**Figure 3.14** – Use of ShapeOp for funicular form finding under fabrication constraints. Demonstrates the effect of combining different constraints: Desired edge length, planarity of faces and desired range of internal face angles. All images show the constrained mesh at equilibrium. From left to right: 1) Shaded rendering. 2) Edge length deviation from desired length. 3) Face angles deviation from desired angle range. 4) Face Planarity Deviation (Red = High deviation, Green = Low deviation). Row 1: The mesh with edge strain constraints. Row 2: The mesh with edge strain and internal mesh face angles constraints. Row 3: The mesh with edge strain and face planarity constraints. Row 4: The mesh with edge strain, internal mesh face angles and face planarity constraints.

### 3.6 Design Process

We believe that ShapeOp can have a considerable impact on a design process in various ways. In interactive modeling tools the graphical user interface often cost non-negligible fraction of the execution time, in particular on large data. Since ShapeOp is built modularly, it can be run independently of any user interface. Also, due to the C/C++ implementation, ShapeOp runs natively and is heavily optimized by compilers and parallelization with OpenMP<sup>4</sup>. ShapeOp can therefore potentially handle huge models.

In ShapeOp, the global and local steps are both numerically stable least-squares problems, implying that the overall method is also stable and robust. Also, many constraints such as the plane constraint only concern the relative arrangement of points and stay satisfied after applying a translation to all points. ShapeOp utilizes this in the global step by implicitly solving for the translations for each constraint independently. This allows for constrained points to move arbitrarily far and greatly increases convergence speed.

Another benefit of ShapeOp is that it is fully open-source, with bindings for many languages including C, C++, C#, Java, and Python. This makes it easy to use ShapeOp from different programming environments, and to extend and adapt the codes according to specific needs.

---

<sup>4</sup>[www.openmp.org](http://www.openmp.org)

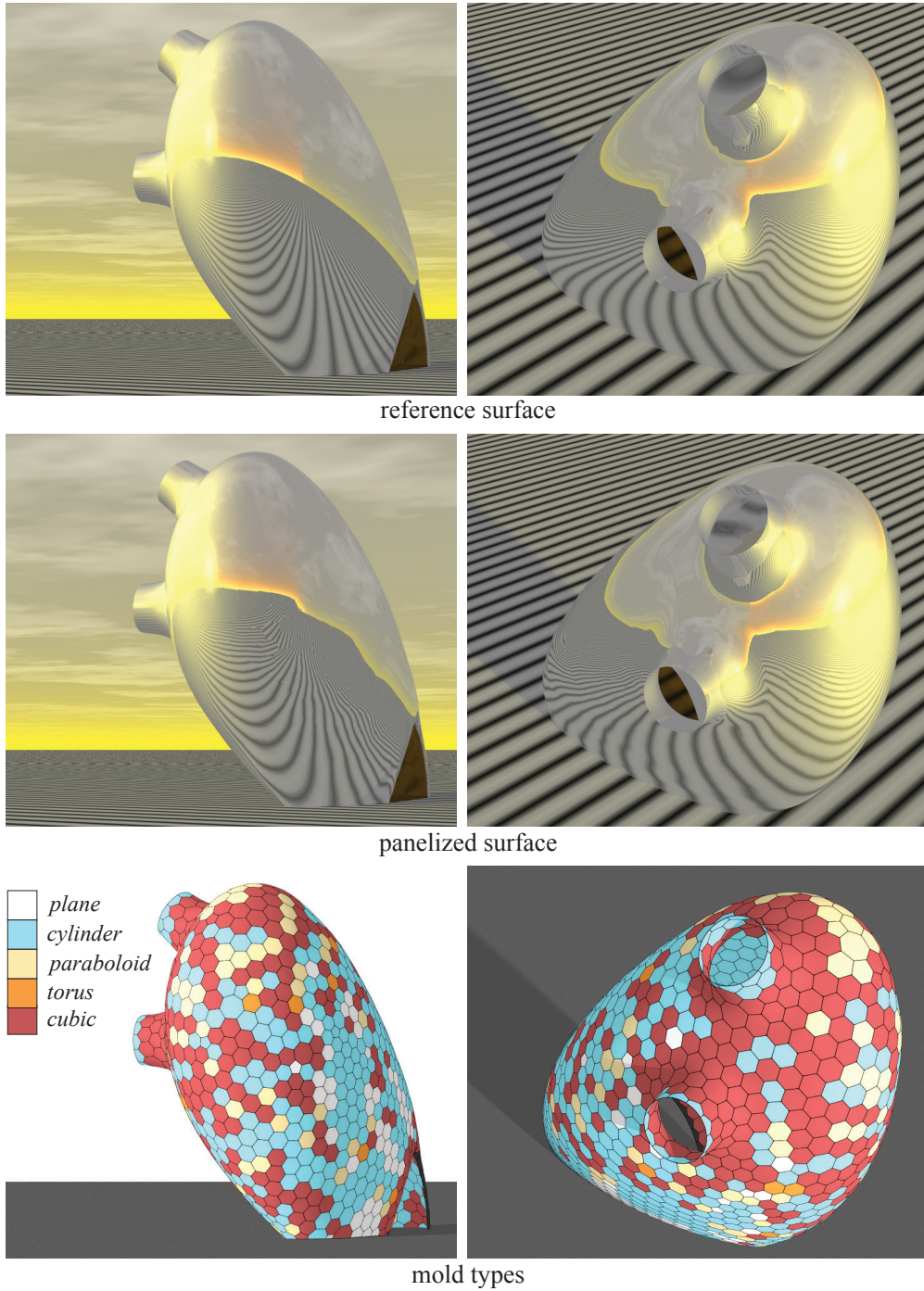
## 4 Cost-Optimized Paneling of Architectural Freeform Surfaces

Paneling an architectural freeform surface refers to an approximation of the design surface by a set of panels that can be manufactured using a selected technology at a reasonable cost, while respecting the design intent and achieving the desired aesthetic quality of panel layout and surface smoothness. Eigensatz and co-workers [EKS<sup>+</sup>10] have recently introduced a computational solution to the paneling problem that allows handling large-scale freeform surfaces involving complex arrangements of thousands of panels. We extend this paneling algorithm to facilitate effective design exploration, in particular for local control of tolerance margins and the handling of sharp crease lines. We focus on the practical aspects relevant for the realization of large-scale freeform designs and evaluate the performance of the paneling algorithm with a number of case studies.

### 4.1 Introduction

Freeform shapes play an increasingly important role in contemporary architecture. Recent technological advances enable the large-scale production of single- and double-curved panels that allow panelizations of architectural freeform surfaces with superior inter-panel continuity compared to planar panels. However, the fabrication of curved panels incurs a higher cost depending on the complexity of the panel shapes, as well as on the employed material and panel manufacturing process (see Table 4.1). This gives rise to the so-called *paneling* task: The approximation of a design surface by a set of panels that can be manufactured using a selected technology at a reasonable cost, while respecting the design intent and achieving the desired aesthetic quality of panel layout and surface smoothness. The paneling task is a key component of the *rationalization* process for architectural freeform designs.

The challenge in paneling architectural freeform surfaces lies in the complex interplay of different objectives related to geometric, aesthetic, and fabrication constraints that



**Figure 4.1** — Given a reference surface (top row), the paneling algorithm produces a rationalization of the the input. The paneling solution (middle row) employs a small set of molds that can be reused for cost-effective panel production (bottom row), while preserving surface smoothness and respecting the original design intent. The shown metal paneling solution is 40% cheaper than the production alternative of using custom molds for each individual panel. Figure 4.11 presents a variety of solutions that achieve cost savings of up to 60%. Figure 4.4 lists the metal cost ratios used.

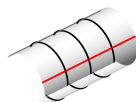
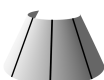


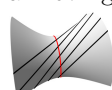
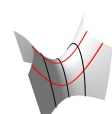
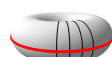


need to be considered simultaneously. In this chapter, we discuss the paneling solution recently introduced in [EKS<sup>+</sup>10], henceforth referred to as the *paneling algorithm*, and focus on the practical aspects relevant for the realization of large-scale freeform designs. We enhance the algorithm to handle spatially adaptive quality thresholds and propose an extension that allows incorporating sharp feature lines. With these new functionalities, the algorithm offers improved control for the architect to adapt the paneling according to the design specifications. We present three case studies to evaluate the performance of the paneling algorithm and provide insights into how the different parameter tradeoffs affect the quality of the results.

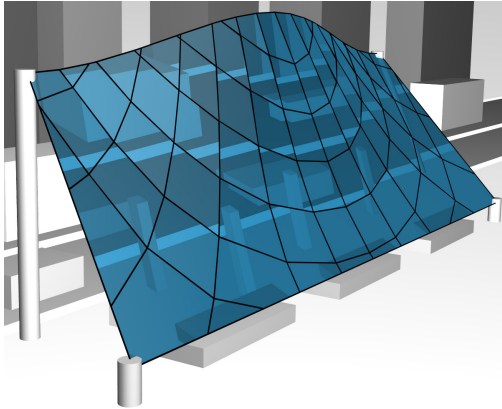
Research related to this work is discussed in Section 2.1, in particular in Subsection 2.1.2. Figure 4.3 compares the original paneling method to other rationalization algorithms. The rest of the chapter is organized as follows: We first classify different available panel types and fabrication processes (Table 4.1). We then formalize the paneling problem as stated in [EKS<sup>+</sup>10] and review the main algorithmic contributions of their paneling solution. Section 4.5 presents our extensions to the existing formulation that allow processing freeform surfaces with sharp feature curves and enable local control of the paneling quality. In Section 4.6, we present three case studies to evaluate the performance of the algorithm.



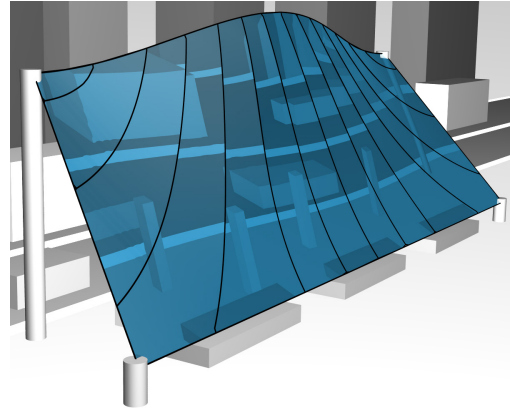
**Figure 4.2** – *Projects involving double-curved panels where a separate mold has been built for each panel. These examples illustrate the importance of the curve network and the existing difficulties in producing architectural freeform structures. (Left: Peter Cook and Colin Fournier, Kunsthhaus, Graz. Right: Zaha Hadid Architects, Hungerburgbahn, Innsbruck.) Figure taken from [EKS<sup>+</sup>10].*

surface types	manufacturing possibilities		
	glass	metal	fibre reinforced concrete/plastic
<u>single curved</u> isometric to the plane, no or little plastic deformation of material			
<i>cylindrical</i> parts of right circular cylinders 	machine for bending and thermal tempering	roll bending machine	configurable mold or custom hot-wire cut foam mold
<i>conical</i> parts of right circular cones 	configurable or custom mold, no thermal tempering	machine or reconfigurable mold	configurable mold or custom hot-wire cut foam mold
<i>general single curved</i> developable surfaces 	custom mold, no thermal tempering		custom hot-wire cut foam mold
<u>double curved</u> usually plastic deformation of material is involved			
<i>general double curved</i> 	custom molds, no thermal tempering of glass	machine or reconfigurable mold	custom molds commonly made of EPS foam
<i>general ruled</i> generated by a moving straight line 	straight lines can be exploited	see above	foam molds can be hot-wire cut
<i>translational</i> 2 families of congruent profiles 	congruent profiles can be exploited		congruent profiles can be exploited
<i>rotational</i> , cf. Figure 4.7 1 family of congruent profiles 			

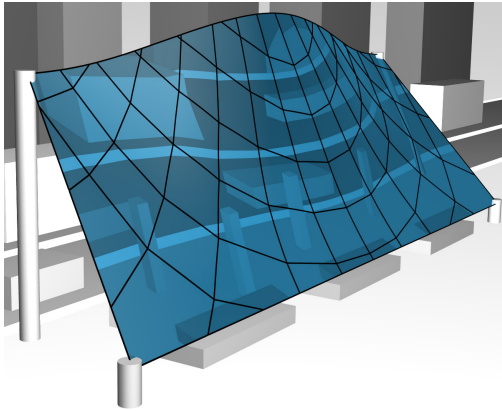
**Table 4.1** – Classification of panel types and typical production processes for common materials in architecture. Although we do not cover all the relevant production processes, this table is for a rough guideline. Planar panels have been left out.



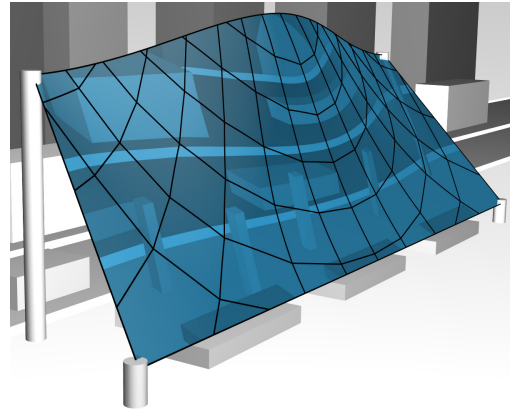
(a) A conical planar quad mesh according to [LPW<sup>+</sup>06] results in a maximum kink angle of 11°.



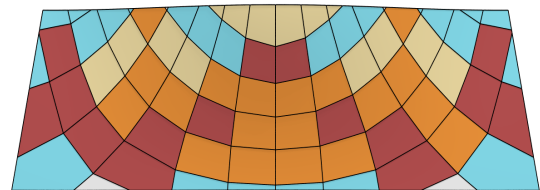
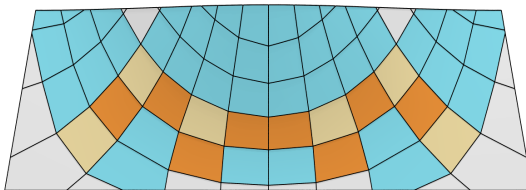
(b) Developable surface strips according to [PSB<sup>+</sup>08] results in a maximum kink angle of 6° between strips.



(c) Paneling solution using 1° kink angle threshold (divergence: 4.7mm; cost: 294).



(d) Paneling solution using 1/4° kink angle threshold (divergence: 1.6mm; cost: 998).

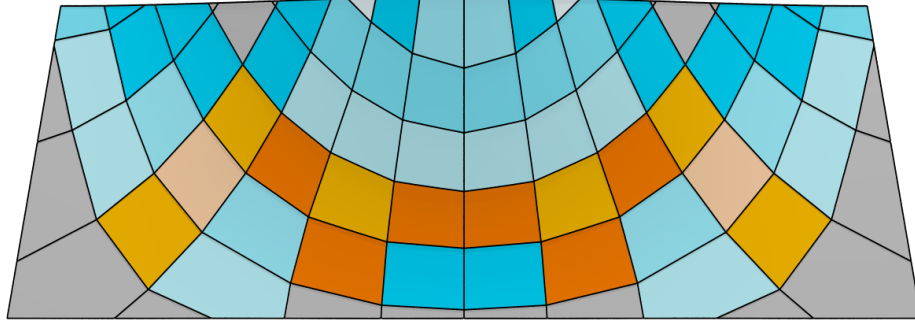


(e) Panels colored by type of corresponding mold. (f) Panels colored by type of corresponding mold.

**Figure 4.3** – Comparison with other rationalization algorithms on a freeform facade design study. (a, b) Rationalization using a planar quad mesh and developable surface strips, respectively. (c-f) Rationalization using the paneling algorithm with 1° and 1/4° kink angle thresholds, shown along with visualization of respective mold types (using glass cost ratios listed in Figure 4.4). A detailed overview of mold reuse for (e) is shown in Figure 4.5.



<i>mold type</i>	plane	cylinder	parab.	torus
<i>cost</i>				
cost per <b>mold</b> depend- ent on type	plane 1	cylinder 1 cylinder 2 cylinder 3 cylinder 4-6 cylinder 7-8	paraboloid 1-3	torus 1 torus 2
cost per <b>panel</b> depend- ent on type	18 panels	16 panels 8 panels 6 panels 4 panels each 1 panel each	2 panels each	2 panels 6 panels

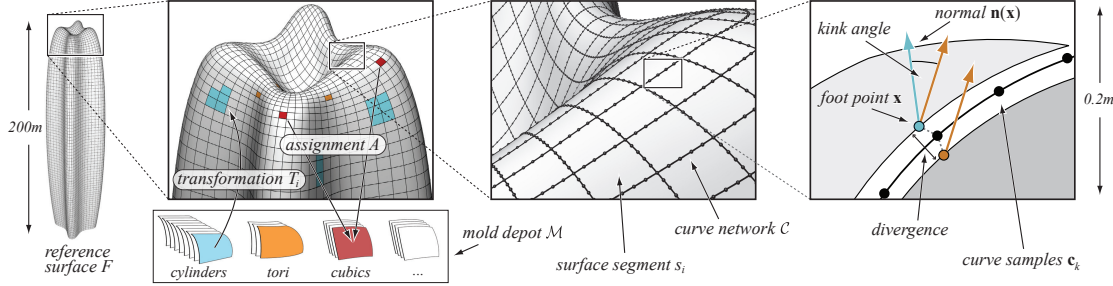


**Figure 4.5** – Illustration of the mold depot and the cost model by means of the example shown in Figure 4.3(e). The colors of panels are saturated according to mold reuse. Figure 4.4 lists the glass cost ratios used for this example.

Mold reuse is a critical cost saving factor. In order to compute paneling solutions with mold reuse in reasonable time one needs to restrict the search space and parameterize panel types using a few parameters only. The paneling algorithm, therefore, uses the restricted panel types paraboloids, tori and cubics instead of the much more general translational, rotational and general double-curved surfaces. Paraboloid, torus, and cubic are defined by 2, 3 and 6 shape parameters, respectively (please refer to the Appendix of [EKS<sup>+</sup>10] for details).

### 4.3 Paneling Architectural Freeform Surfaces

We review both the specification of the paneling problem and the optimization approach presented by Eigensatz and coworkers. For a more detailed description, in particular with respect to mathematical and algorithmic aspects, we refer the reader to Section 4.4 and [EKS<sup>+</sup>10].



**Figure 4.6** – Terminology and variables used in the paneling algorithm. The reference surface  $F$  and the initial curve network  $C$  are given as part of the design specification. The optimization solves for the mold depot  $M$ , the panel-mold assignment function  $A$ , the shape parameters of the molds, the alignment transformations  $T_i$ , and the curve network samples  $c_k$ . Figure taken from [EKS<sup>+</sup>10].

#### 4.3.1 Problem Specification

Let  $F$  be a given input freeform surface, called *reference surface*, describing the shape of the design. The goal is to find a collection of *panels*, such that their union approximates the reference surface. Since the quality of the approximation strongly depends on the position and tangent continuity across panel boundaries, Eigensatz and coworkers identify two quality measures (see Figure 4.6):

- **divergence:** quantifies the spatial gap between adjacent panels and,
- **kink angle:** measures the jump in normal vectors between adjacent panels.

While divergence is strongly related to the viability of a paneling solution, the kink angles influence the visual appearance, since they are related to reflections. Hence one can allow higher kink angles in areas not or only barely visible to an observer. We will elaborate on this possibility in Sections 4.5.2 and 4.6.2.

The intersection curves between adjacent panels are essential for the visual appearance of many designs (see Figure 4.2) and typically reflect the structure of the building, as they often directly relate to the underlying support structure. An initial layout of these curves is usually provided by the architect or engineer as an integral part of the design. While small deviations are typically acceptable in order to improve the paneling quality, the final solution should stay faithful to the initial curve layout and reproduce the given

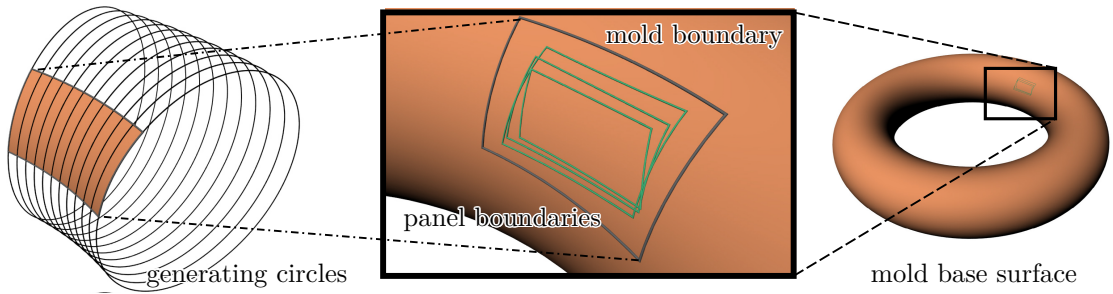


pattern as good as possible by the intersection lines of adjacent panels. The collection of all panel boundary curves (strictly speaking panel intersection curves) forms the *curve network*, which splits the given input freeform surface into *segments*. Each segment, in general polygonal, of the curve network has to be covered by a panel.

The *paneling problem* is formulated as follows: Approximate a given freeform surface  $F$  by a collection of panels of selected types such that pre-defined thresholds on divergence and kink angle are respected, the initial curve network is reproduced as good as possible, and the total production cost is minimized. The production cost of a panelization comprises the following terms: the production cost of each employed mold and the cost of producing each panel from its assigned mold (see Figure 4.4 for two typical cost sets and Figure 4.5 for an illustration).

#### 4.3.2 Paneling Algorithm

A paneling solution can be computed using the optimization algorithm described in [EKS<sup>+</sup>10]. This algorithm takes as input the reference surface  $F$ , the initial curve network, and global thresholds on maximal kink angle and divergence, along with a permitted deviation margin of the final paneled surface from the reference surface. As output, the algorithm computes the parameters that determine the shape of the fabrication molds and the alignment transformations that position the panels in space. These parameters are computed in such a way that the reference surface is approximated as good as possible, while the kink angle and divergence thresholds are satisfied everywhere. At the same time, the cost of fabrication is minimized by favoring panels that are geometrically simple and thus cheaper to manufacture wherever possible, and maximizing the amount of mold reuse.



**Figure 4.7** – *Example of mold reuse. Panel boundary curves are in general not congruent. However, several panels may be closely grouped together on the same mold base surface. In that case the same mold or machine configuration, which embraces all affected panels, may be used to manufacture the panels. This figure further illustrates how the congruent profiles of a rotational or translational surface, in this case the circles generating a torus, can be exploited for mold fabrication.*

In order to achieve these conflicting goals, the paneling optimization is formulated as

a mixed discrete/continuous optimization that simultaneously explores many different paneling solutions (see Section 4.4 for details). From all these different alternatives, the solution of minimal overall fabrication cost is selected that satisfies the kink angle and divergence thresholds. An essential ingredient in this optimization is controlled deviation of the paneling from the initial design surface. By allowing the curve network to move away from the reference surface, panels can fit together with smaller kink angles and divergence, simpler and thus cheaper panels can be used in certain regions, and the amount of reuse of molds can be increased. Figure 4.8 demonstrates the effectiveness of the discrete optimization presented by [EKS<sup>+</sup>10] on an illustrative example, comparing different techniques to enable mold reuse.

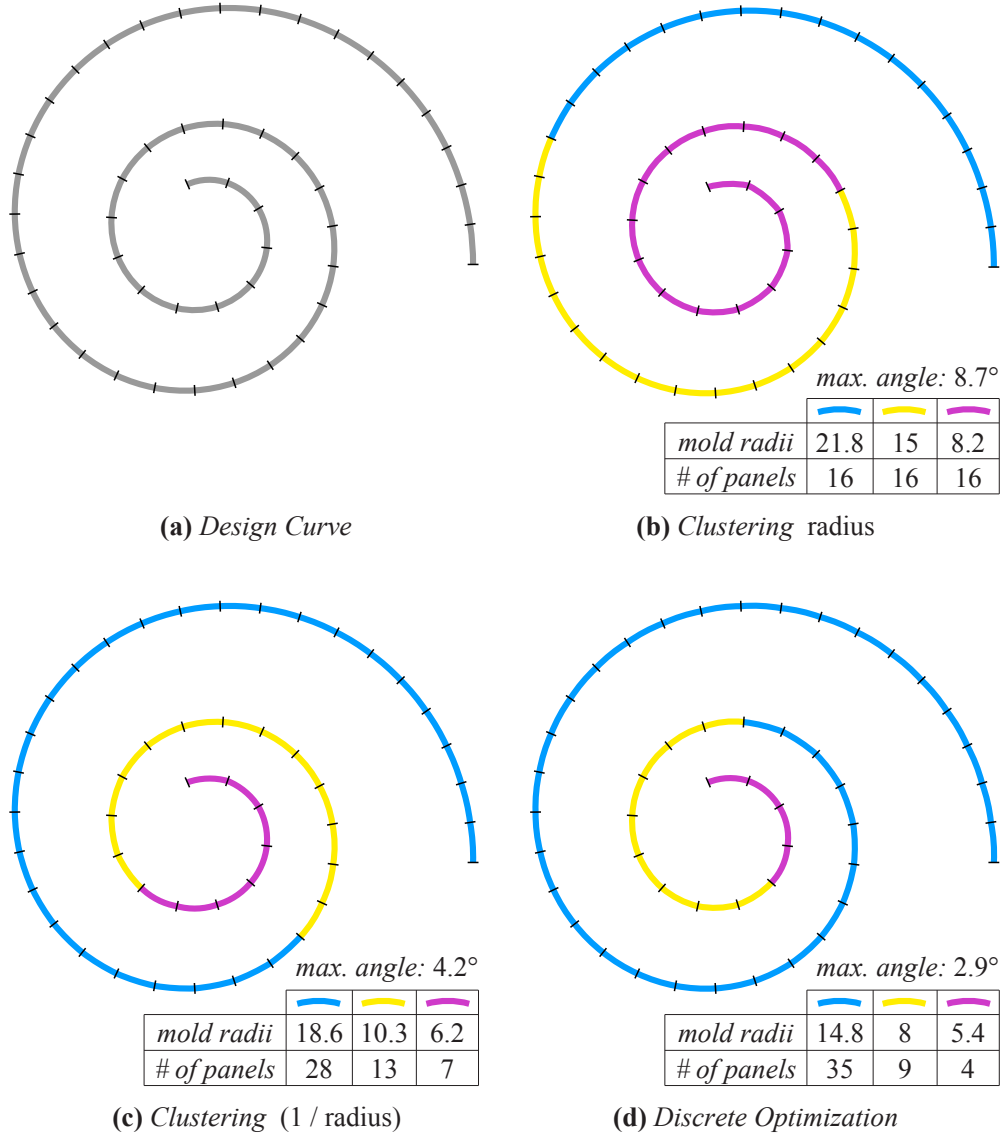
The results shown in [EKS<sup>+</sup>10] include solutions to the paneling problem for large-scale architectural freeform designs that often consist of thousands of panels. Typically, these paneling solutions consist of patches of flat, single and double curved panels as shown in Figure 4.3, therefore partly generalizing the approaches introduced in [LPW<sup>+</sup>06] and [PSB<sup>+</sup>08] to include double curved panels.

The main innovations of the paneling algorithm can be summarized as follows:

- Given a table of mold and panel production costs, the paneling algorithm computes a panelization with minimal cost while meeting predefined quality requirements.
- The algorithm is adaptable to numerous production processes and materials.
- The possibility to explore diverse quality requirements and cost tables provides valuable information to guide design decisions.
- The rationalized 3D models produced by the algorithm may be used for visual inspection, prototype panel manufacturing, quality control, and the final production of freeform surfaces.
- Interference with the architects design intent is minimized.

The original paneling algorithm provides a general framework and is extensible in various ways. We first provide more details of the original method in 4.4, then propose and investigate two specific extensions in Section 4.5 and discuss further extension possibilities in Section 6.2.





**Figure 4.8** – Illustrative comparison of different techniques for mold reuse. The curve should be approximated with circle arcs of varying radii. This can be understood as a simple paneling with cylinders of varying radii, where the figure shows an orthogonal cross section. The input design curve shown in (a) consists of nicely aligned circle arcs with decreasing radii from 25 to 5. The method shown in (b) clusters these radii (using  $k$ -means clustering) to obtain 3 molds and assigns the best mold to each segment. The colors indicate the segments sharing the same mold. The method shown in (c) does the same, but performs a clustering of  $(1/\text{radius})$  instead of clustering the radius itself, which is a much better distance approximation for cylinders as shown in [EKS<sup>+</sup>10] and therefore the maximal kink angle is already much lower compared to (b). The method shown in (d) performs the full discrete optimization presented in [EKS<sup>+</sup>10] and leads to an even better mold depot that enables a paneling with only 3 molds but very low kink angles. The differences presented on this schematic example become even more prominent if more complex surfaces and/or panel types are involved.

### 4.4 Formalization of the Paneling Algorithm

This section is a summary of the original paneling algorithm, providing the necessary details for completeness of this chapter. We first describe the paneling problem and then the algorithmic approach of [EKS<sup>+</sup>10].

#### 4.4.1 Problem

In the following, we formalize the paneling problem. We discuss input, goal, output, cost and representation.

##### Input

The input to the paneling algorithm consists of a freeform surface  $F$  and an initial curve network  $\mathcal{C}$  on  $F$ , both describing the design intent. Each part of  $F$ , bounded by curves of  $\mathcal{C}$ , defines a segment  $s_i$ , which will be approximated by a panel  $P_i$ . Additionally, the cost  $c(M_k)$  for producing a mold  $M_k$  of a specific type, and the cost  $c(M_k, P_i)$  for producing a panel  $P_i$  from  $M_k$ , has to be provided for each supported panel type.

##### Goal

The goal of the paneling algorithm is to find a collection of panels  $\mathcal{P}$  such that their union approximates the input surface  $F$  and their boundaries follow  $\mathcal{C}$ . The algorithm minimizes the cost of producing  $\mathcal{P}$ , while respecting the thresholds on divergence and kink angles provided by the user.

##### Output

The output of the paneling algorithm contains a collection of panels  $\mathcal{P} = \{P_1, \dots, P_n\}$  and a set of Molds  $\mathcal{M} = \{M_1, \dots, M_m\}$ , with  $m \leq n$ .  $\mathcal{M}$  represents the mold depot, which can be used to produce all panels in  $\mathcal{P}$ . The algorithm computes an explicit panel-mold assignment function  $A : [1, n] \rightarrow [1, m]$  mapping each panel  $P_i \in \mathcal{P}$  to its mold  $M_k \in \mathcal{M}$ , as well as a set of rigid transformations  $\mathcal{T} = \{T_1, \dots, T_n\}$ , where  $T_i$  describes the global placement of panel  $P_i$  in the final paneling.

### Cost

Given the cost  $c(M_k)$  of the mold  $M_k$  and  $c(M_k, P_i)$  of the panel  $P_i$  made with  $M_k$ , the total production cost of the final paneling is measured by

$$\text{cost}(F, \mathcal{P}, \mathcal{M}, A) = \sum_{k=1}^m c(M_k) + \sum_{i=1}^n c(M_{A(i)}, P_i). \quad (4.1)$$

### Representation

The curve network  $\mathcal{C}$  is represented as an explicit network of polygonal lines that can deviate from the input surface  $F$  during optimization. Each vertex  $\mathbf{c}_l$  of  $\mathcal{C}$  is represented by a scalar offset along the normal of  $F$ , which reduces the number of variables significantly, while preserving the design intent. This explicit representation of  $\mathcal{C}$  avoids numerical instabilities when intersecting well-aligned neighboring panels, simplifies the formalization of the surface fitting and kink angle minimization and enables cheaper solutions with better quality (smaller kink angles and divergence). A mold  $M_k$  is represented by a set of parameters fully describing the mold surface (see Appendix of [EKS<sup>+</sup>10] for details). Note that a panel  $P_i$  produced from  $M_k$  is usually only a part of the full mold surface (see Figure 4.7).

#### 4.4.2 Algorithm

The number and types of molds and the assignment function are discrete variables, while the other variables are continuous. This makes the paneling problem, as formalized above, a mixed discrete/continuous optimization problem. The paneling algorithm tackles this difficult optimization problem by alternating continuous (Section 4.4.3) and discrete (Section 4.4.4) optimization steps. Section 4.4.5 describes how these two optimizations are combined to yield the full paneling algorithm.

#### 4.4.3 Continuous Optimization

The continuous optimization reduces divergence and kink angles, while keeping the number and types of molds and the assignment function fixed. The optimization is formalized as a non-linear least-squares problem consisting of a weighted sum of the five least-squares objective functions, respectively energies, described in the following paragraphs.

The **surface fitting** energy minimizes the distance of the curve network  $\mathcal{C}$  to the input surface  $F$ . The energy measures the sum of distances between  $\mathbf{c}_l$  and the points  $\mathbf{f}_l$  on  $F$  closest to  $\mathbf{c}_l$ .

$$E_{\text{fit}} = \sum_{l=1}^L \|\mathbf{c}_l - \mathbf{f}_l\|^2 \quad (4.2)$$

The **divergence** energy is approximated by the sum of distances between the curve network vertices  $\mathbf{c}_l$  and the points  $\mathbf{x}_{i(l)}$  and  $\mathbf{x}_{j(l)}$ , on the adjacent aligned mold surfaces  $M_{i(l)}^*$  and  $M_{j(l)}^*$ , closest to  $\mathbf{c}_l$ .

$$E_{\text{div}} = \sum_{l=1}^L \|\mathbf{c}_l - \mathbf{x}_{i(l)}\|^2 + \|\mathbf{c}_l - \mathbf{x}_{j(l)}\|^2 \quad (4.3)$$

The **kink angle** energy achieves tangent continuity by minimizing the sum of differences between the normal vector  $\mathbf{n}(\mathbf{x}_{i(l)})$  of the aligned mold  $M_{i(l)}^*$  at  $\mathbf{x}_{i(l)}$ , and  $\mathbf{n}(\mathbf{x}_{j(l)})$  defined analogously (Figure 4.6).

$$E_{\text{kink}} = \sum_{l=1}^L \|\mathbf{n}(\mathbf{x}_{i(l)}) - \mathbf{n}(\mathbf{x}_{j(l)})\|^2 \quad (4.4)$$

The **curve fairness** energy minimizes undulations in the curve network  $\mathcal{C}$ . Tangential undulations are completely avoided by the choice of representing the curve network vertices  $\mathbf{c}_l$  by a scalar offset  $d_l$  from the initial curve network, along the normal of  $F$ .  $\mathcal{I}_{\mathcal{C}}$  is the set of vertex index pairs connected by an edge in the curve network  $\mathcal{C}$ .

$$E_{\text{fair}} = \sum_{(j_1, j_2) \in \mathcal{I}_{\mathcal{C}}} (d_{j_1} - d_{j_2})^2 \quad (4.5)$$

The **panel centering** energy keeps all panels produced by the same mold close to each other on the mold surface. This reduces the size of the part of the mold surface that has to be manufactured for producing the panels.  $\mathbf{b}_i$  is an approximation of the barycenter of the segment  $s_i$ .  $\mathbf{b}_i$  is computed as the average of all adjacent vertices.  $\mathbf{p}_i$  is the projection of  $\mathbf{b}_i$  onto the normal of the mold surface at the center of the aligned mold  $M_i^*$ .

$$E_{\text{cen}} = \sum_{i=1}^n \|\mathbf{b}_i - \mathbf{p}_i\|^2 \quad (4.6)$$

### Global optimization

The above least-squares energies are weighted and summed up to form the global objective function  $E$ , which is minimized using a Gauss-Newton solver.

$$E = \alpha_{\text{fit}} E_{\text{fit}} + \alpha_{\text{div}} E_{\text{div}} + \alpha_{\text{kink}} E_{\text{kink}} + \alpha_{\text{fair}} E_{\text{fair}} + \alpha_{\text{cen}} E_{\text{cen}}. \quad (4.7)$$

Unless stated otherwise, the following weights are used:  $\alpha_{\text{fit}} = 1$ ,  $\alpha_{\text{div}} = 1000$ ,  $\alpha_{\text{fair}} = 1$ ,  $\alpha_{\text{cen}} = 10$ . The paneling system sets  $\alpha_{\text{kink}} = (\epsilon/\delta)^2 \alpha_{\text{div}}$ , where  $\epsilon$  is the threshold on divergence and  $\delta$  the threshold on kink angles.

#### 4.4.4 Discrete Optimization

The discrete optimization finds a mold depot  $\mathcal{M}$  and assignment function  $A$  which minimize the production cost of panels and keep divergence and kink angles below the given thresholds. The optimization first computes a set of candidate molds  $\mathcal{M}'$ , from which it then chooses a cost-minimizing subset  $\mathcal{M} \subseteq \mathcal{M}'$  respecting the thresholds.  $\mathcal{M}'$  is initialized with the molds of the current mold depot, and enriched by five molds per segment, which are the result of fitting a mold of each type to the segment. To simplify the fitting, each curve network vertex stores the average normal of all closest points on incident panels of the current paneling solution. This allows to use faster, localized versions of the divergence, kink angle and panel centering energies (Equations 4.3, 4.4, 4.6) for fitting.

#### Set cover

Let  $\mathcal{S}_k = \{s_{k_1}, \dots, s_{k_l}\}$  be the set of surface segments that can be approximated by  $M_k$  while respecting the thresholds. Computing the sets  $\mathcal{S}_k$  requires a non-linear alignment of each candidate mold with every segment, which is a critical performance bottleneck. To reduce the computational effort, the paneling algorithm evaluates approximate alignment distances in a 6D euclidian space, which allows to reject 95-99% of the segments before an expensive, non-linear alignment (see Appendix of [EKS<sup>+</sup>10]). The production cost of  $\mathcal{S}_k$  is given by  $c(\mathcal{S}_k) = c(M_k) + |\mathcal{S}_k| c(M_k, P_*)$ . The optimal mold depot is given by the cheapest set of molds  $M_k$  which covers all segments  $\mathcal{S} = \{s_1, \dots, s_n\}$ . The problem of finding this mold depot is reminiscent to the classical *weighted set cover* problem [Joh74]. This problem is known to be NP-hard, therefore the paneling algorithm uses the approximation algorithm presented in [Fei98], which has the best possible approximation ratio  $\log n$  of any polynomial-time algorithm. While the problem is not equivalent to weighted set cover, the proof of the approximation algorithm generalizes directly to it.

### Algorithm

The discrete optimization algorithm requires a notion of *efficiency*  $\phi(\mathcal{S}_k, \mathcal{S}')$  of a set  $\mathcal{S}_k$  with respect to the yet uncovered segments  $\mathcal{S}'$ . In the discrete part of the paneling algorithm, the efficiency is defined as  $\phi(\mathcal{S}_k, \mathcal{S}') = |\mathcal{S}_k|/c(\mathcal{S}_k)$ , where the size and cost of  $\mathcal{S}_k$  considers only segments of  $\mathcal{S}'$ . The algorithm iteratively finds covering sets  $\sigma$ , and keeps track of the so far unused sets  $\sigma'$  (pseudo-code taken from [EKS<sup>+</sup>10]):

```

 $\sigma \leftarrow \emptyset, \sigma' \leftarrow \{\mathcal{S}_1, \dots, \mathcal{S}_{|\mathcal{M}'|}\}, \mathcal{S}' \leftarrow \mathcal{S}_1 \cup \dots \cup \mathcal{S}_{|\mathcal{M}'|}$ 
while  $\mathcal{S}' \neq \emptyset$ 
    eval.  $\phi(\mathcal{S}_k, \mathcal{S}') \quad \forall \mathcal{S}_k \in \sigma' \quad \text{update efficiencies}$ 
     $\mathcal{S}_i \leftarrow \arg \max_{\mathcal{S}_k \in \sigma'} \phi(\mathcal{S}_k, \mathcal{S}') \quad \text{set with max. efficiency}$ 
     $\sigma \leftarrow \sigma \cup \{\mathcal{S}_i\} \quad \text{add to covering sets}$ 
     $\mathcal{S}' \leftarrow \mathcal{S}' - \mathcal{S}_i, \sigma' \leftarrow \sigma' - \{\mathcal{S}_i\} \quad \text{remove covered segments}$ 
end

```

After running this algorithm, segments that cannot be covered by any mold candidate get assigned the best-fit cubic mold. Segments covered by multiple molds get assigned the cheapest of them.

#### 4.4.5 Alternating Optimization

The paneling algorithm start with thresholds  $\epsilon' = \epsilon + 10mm$  and  $\delta' = \delta + 5^\circ$ , then iteratively reduces them until  $\epsilon' = \epsilon$  and  $\delta' = \delta$ . The algorithm starts with a single plain as a mold depot, and first performs a re-initialization step, then iterates the steps listed below ten times, and finishes with a discrete step using  $(\epsilon, \delta)$ .

1. Discrete optimization
2. Continuous optimization
3. Re-initialization
4. Reduce  $\epsilon'$  and  $\delta'$

The re-initialization step first assigns a mold of the cheapest type that satisfies  $(\epsilon', \delta')$  to each segment that did not satisfy  $(\epsilon, \delta)$ , followed by a continuous optimization step. This reverses unsuccessful assignments from the previous discrete step.

## 4.5 Extensions

In this section we discuss algorithmic extensions to the method of Eigensatz and coworkers [EKS<sup>+</sup>10] that broaden its applicability.

### 4.5.1 Sharp Features

The algorithm introduced by Eigensatz and coworkers assumes that the input reference surface is smooth everywhere. Sharp feature lines, however, are used in architectural freeform designs to highlight strong characteristic features and to enhance the visual appeal of a design.

We therefore propose an extension of the paneling algorithm to incorporate sharp features. Sharp feature lines can either be specified by the designer as specially marked lines of the initial curve network, or automatically computed by detecting sharp creases on the design surface. To support sharp features we adapt the original paneling algorithm such that

- kink angle thresholds are not applied along the curves describing sharp features and
- the tangent continuity between two panels on opposite sides of a sharp feature is not optimized.

Figure 4.14 demonstrates how this extension enables paneling freeform surfaces with sharp features.

### 4.5.2 Adaptive Control of Paneling Quality

The paneling algorithm introduced in [EKS<sup>+</sup>10] guarantees compliance with user-specified tolerance thresholds on divergence and kink angle. These thresholds are specified globally for the entire surface. In practice, however, the quality requirements might vary for different regions of the design. For regions not visible from certain view-points, for example, higher kink angles might be acceptable to reduce manufacturing cost. We therefore extend the original paneling algorithm to optimize the paneling quality with respect to a spatially adaptive importance function on the design surface.

As shown in Figure 4.10 this importance function can, for example, be computed using a visibility calculation that computes the visibility for every point on the design surface, if the design is viewed from a path or street around the building. This importance function is then an additional input to our extended paneling algorithm to

- adaptively specify a separate kink angle threshold for every point on the curve

network and

- focus the tangent continuity optimization on important regions.

Figures 4.10-4.13 demonstrate how this adaptive quality control directs the use of expensive panels towards regions where they are needed most, leading to an improved paneling quality at similar or lower costs compared to globally specifying thresholds. Achieving the same quality at the important regions with the original paneling algorithm using global thresholds requires a much more expensive paneling. The same technique can be used to adaptively control the divergence or the deviation from the original design surface.

## 4.6 Case Studies

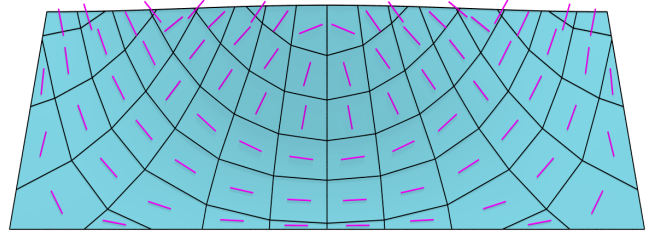
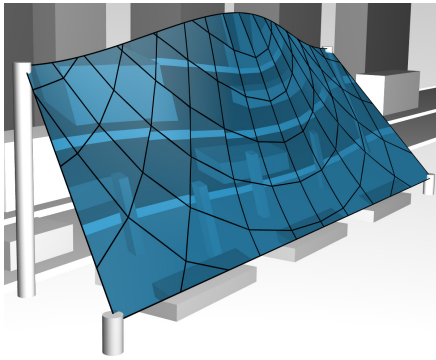
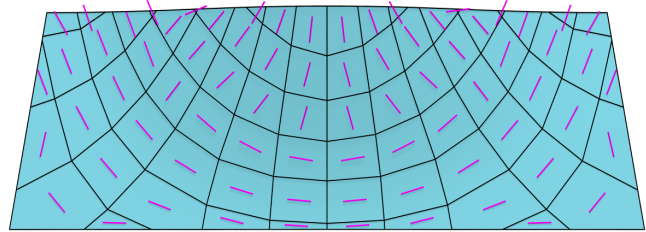
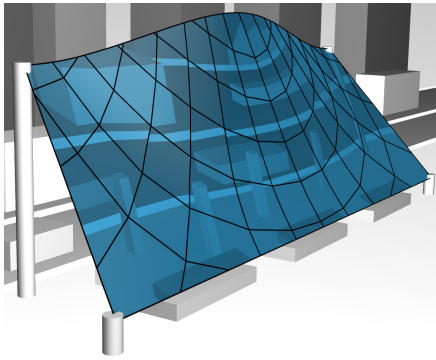
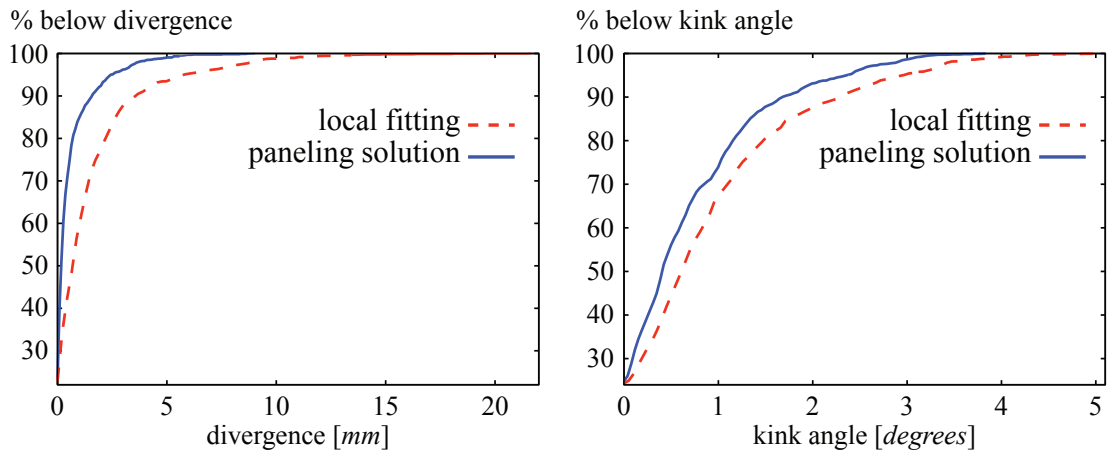
In this section we demonstrate the performance of the paneling algorithm on three case studies. Specifically we compare our solutions with state-of-the-art rationalization alternatives, study the preservation of sharp features, and compare the cost trade-offs for global kink angle specifications versus spatially adapted ones.

### 4.6.1 Facade Design Study

We compare several rationalization possibilities for a freeform facade. For this case study we use glass mold cost ratios as listed in Figure 4.4.

Figure 4.3a shows a rationalization result using a conical planar quad mesh, which implies very favorable properties for simplifying the substructure, cf. [LPW<sup>+</sup>06, PLW<sup>+</sup>07]. Naturally this approach leads to a faceted result with kink angles up to 11°. A further option makes use of the close relation between planar quad meshes and developable strip models ([PSB<sup>+</sup>08]): Refining the planar quad mesh in one direction and keeping the faces planar leads to a rationalization using single-curved strips. Clearly this results in a much smoother representation of the surface as can be seen in Figure 4.3b (maximum 6° kink angle), while one could still make use of a planar quad mesh for the substructure. The deformation of glass to general single-curved panels, however, requires molds to be built, a possibility that was ruled out because of budgetary issues. Therefore the paneling algorithm was used to proof feasibility for the competition, making use of cylindrical panels only. The superiority of such a restricted paneling solution to results that are achievable using local fitting of cylinders is documented in Figure 4.9. Figure 4.3 compares further paneling solutions with respect to cost and paneling quality, making use of the complete set of mold types.



(a) *Local fitting of cylinders.*(b) *Paneling solution.*(c) *Cumulative histograms of divergence and kink angles for the above solutions.*

**Figure 4.9** – The paneling algorithm restricted to cylindrical panels. Here we compare a result on the Facade Design Study computed using simple local fitting of cylinders (a) to a paneling solution using only cylinders (b). For both results we show the axis directions of cylinders colored in magenta and the cumulative histograms of resulting divergences and kink angles.

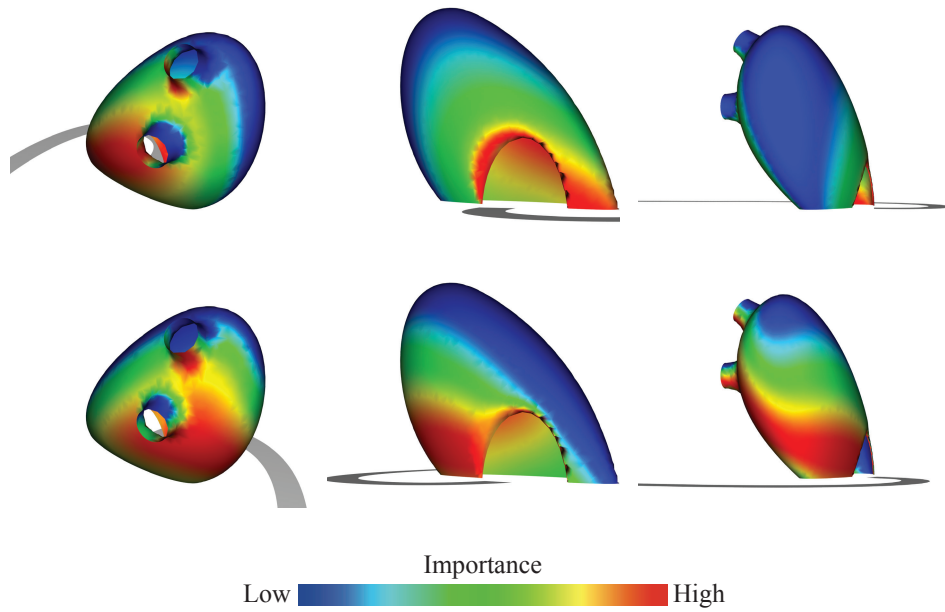
### 4.6.2 Skipper Library

Initially issued by Texxus, the skipper library is a feasibility study also picked up by Formtexx for freeform metal cladding. The case study demonstrates our extension of the paneling algorithm allowing adaptive control of the paneling quality, as well as the ability of the paneling algorithm to handle arbitrary panel layouts. The presented panel layout was created using the dual mesh of a circle packing mesh (cf. [SHWP09]), which leads to a panel layout consisting mainly of hexagonal panels combined with a torsion free support structure. Our motivation to adaptively control the paneling quality is given by the following:

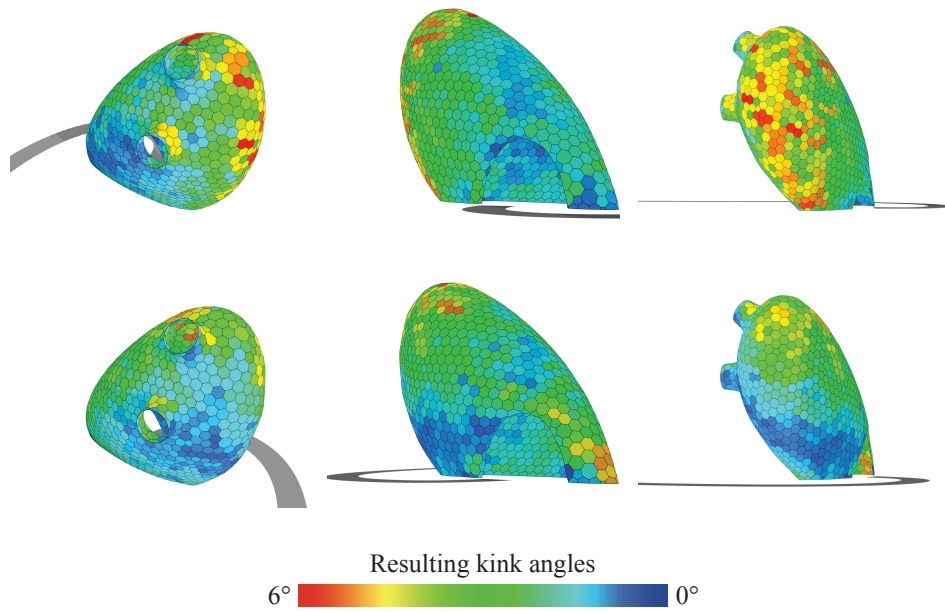
Due to various constraints imposed by surrounding buildings, restricted access paths, neighboring trees and foliage, different sections of architectural buildings have different visibility. This can be exploited to reduce the manufacturing cost of such buildings by allowing larger kink angles in less visible regions. As described in Section 4.5, we generalize the paneling algorithm proposed in [EKS<sup>+</sup>10] to allow spatially variable kink angle specifications as opposed to a global maximum kink angle threshold. Figures 4.10-4.13 compare the results on manufacturing cost for a global threshold versus two spatially adapted threshold specifications. The local importance functions are computed based on visibility of the reference surface when moving along the specified access paths (see Figure 4.10). For this case study we use metal mold cost ratios as listed in Figure 4.4. The middle row in Figure 4.1 shows a paneling solution with 1° global kink angle threshold.

### 4.6.3 Lissajous Tower

Lissajous Tower is an example skyscraper specifically created for illustrating our extension to the paneling algorithm for handling sharp features. The surface contains large nearly flat and single-curved parts as well as small highly curved parts, which can not be approximated by cylinders within realistic tolerances. Figure 4.14 compares two paneling solutions produced by the paneling algorithm with maximum kink angle thresholds of 1° and 3°, respectively. While both solutions preserve the characteristic sharp feature line of the design, the production cost is significantly reduced (by 40%) for a slight relaxation in the maximum kink angle constraint. For this case study we use glass mold cost ratios as listed in Figure 4.4.

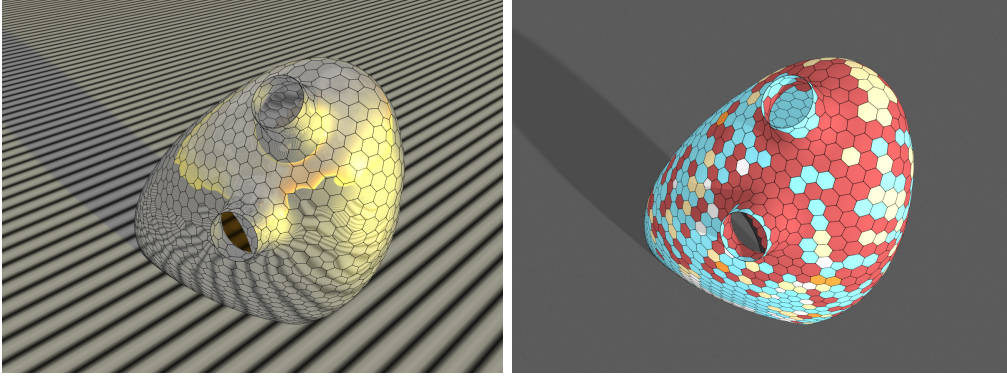


(a) Spatially adaptive importance functions computed based on visibility from path 1 (top row) and path 2 (bottom row). These importance functions are used for paneling solutions as shown in 4.10(b) and Figures 4.11-4.13 (b) and (c), respectively.

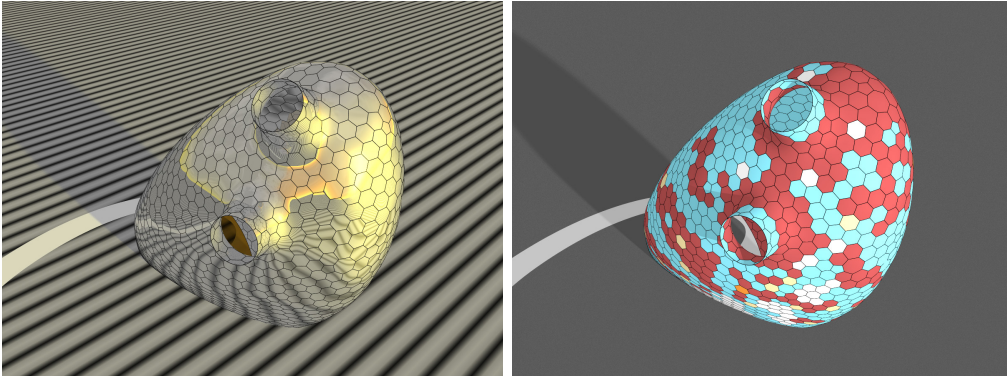


(b) Kink angles of two paneling solutions (top and bottom rows) using adaptive thresholds based on the two importance functions shown in 4.10(a). Further renderings of the results are shown in Figures 4.11-4.13.

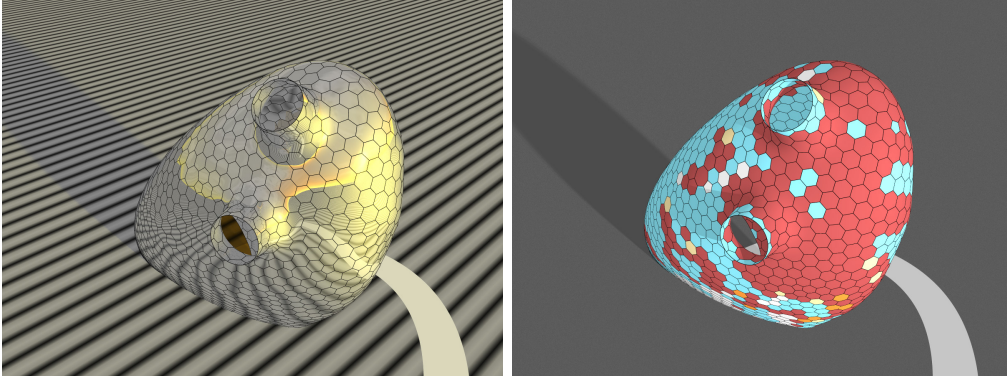
**Figure 4.10** – Adaptive quality control.



(a) Paneling solution with kink angle thresholds specified globally over the surface.



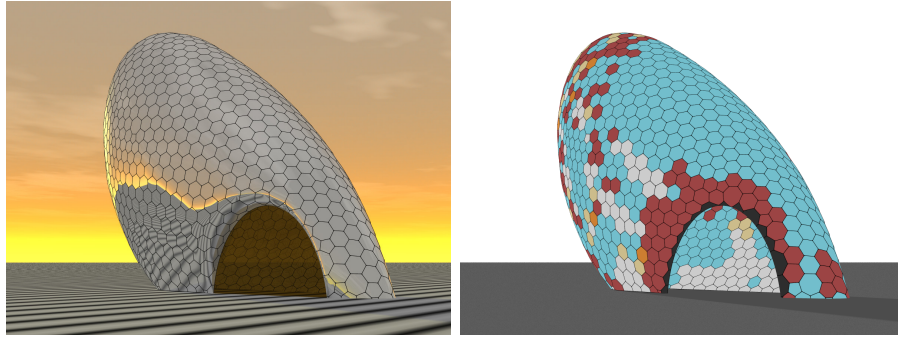
(b) Paneling solution with spatially adaptive kink angle thresholds.



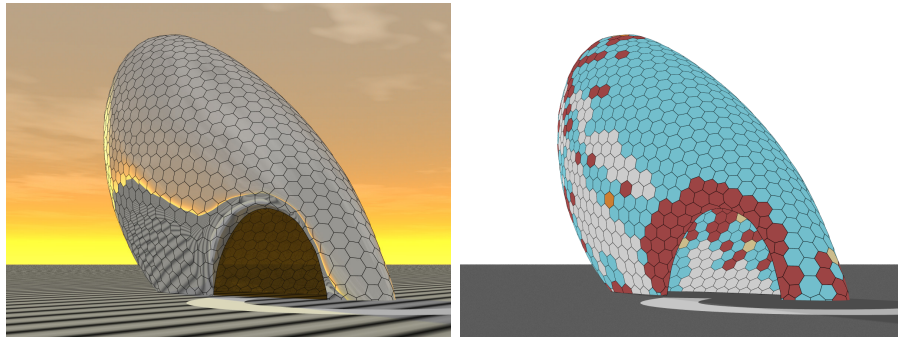
(c) Paneling solution with another set of spatially adaptive kink angle thresholds.

**Figure 4.11** – Effect of global vs spatially varying kink angle specifications on the Skipper Library dataset. Paneling solutions using a global kink angle specification (a) and using adaptive kink angle thresholds computed based on the extent of visibility while moving along the indicated ground paths (b, c). Left column images show the reflection lines on paneled surfaces, while right column images show the mold types for individual panels (color convention same as in Figure 4.1). Figures 4.12 and 4.13 show the same solutions from two other views. Figure 4.10 shows the spatially varying kink angle thresholds used in (b) and (c).

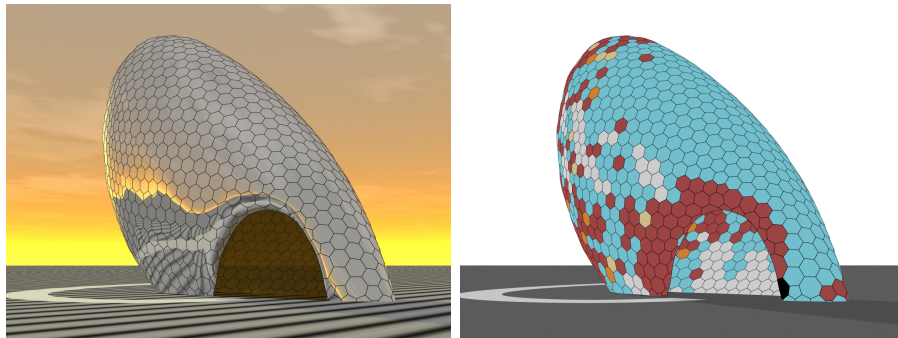




(a) Paneling solution with kink angle thresholds specified globally over the surface.



(b) Paneling solution with spatially adaptive kink angle thresholds.



(c) Paneling solution with another set of spatially adaptive kink angle thresholds.

(a) global cost: 5946

<i>molds</i>	-	38	15	2	119	32
<i>panels</i>	102	622	84	11	349	32

divergence: 6mm

max angle: 3°

(b) path 1 cost: 5810

-	73	8	1	169	22
152	683	17	5	321	22

divergence: 6mm

max angle: 1°-6° (adaptive)

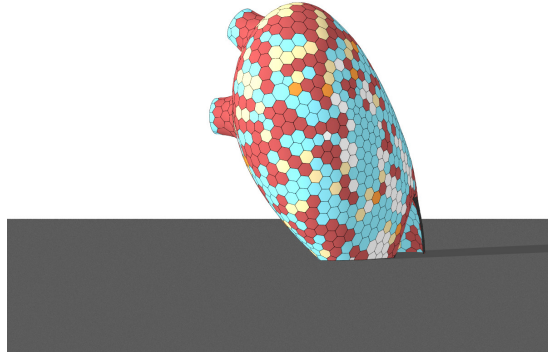
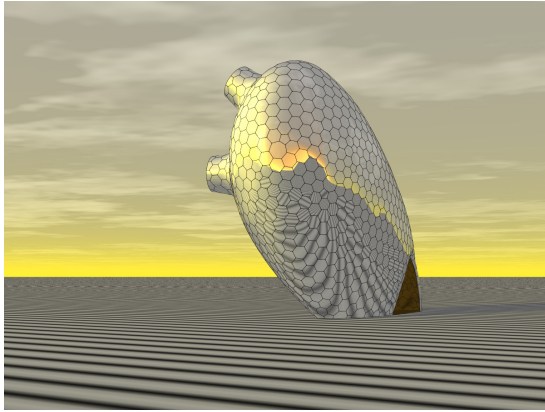
(c) path 2 cost: 6265

-	45	7	5	191	15
97	631	17	13	427	15

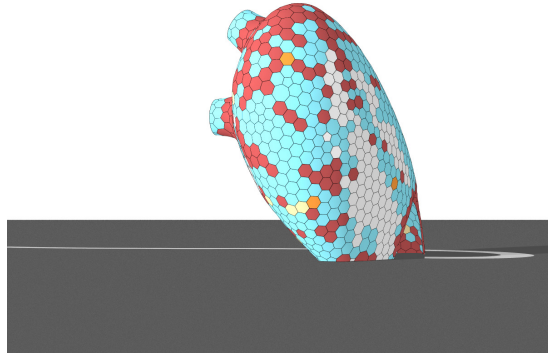
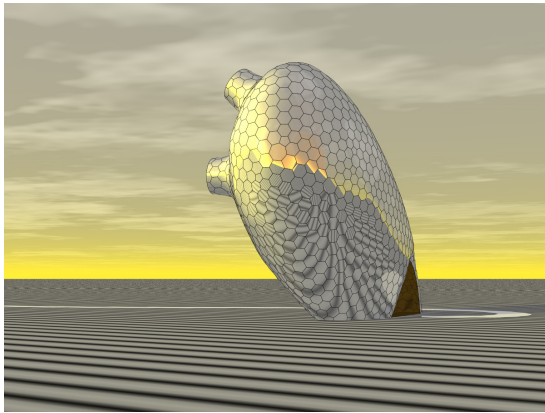
divergence: 6mm

max angle: 1°-6° (adaptive)

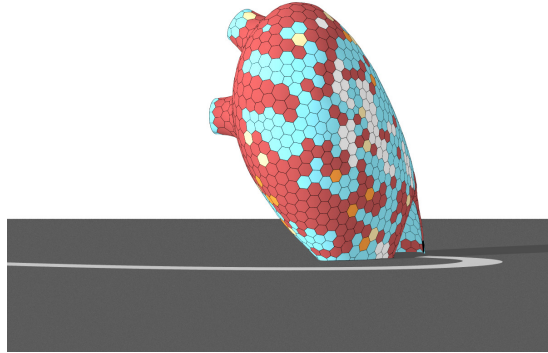
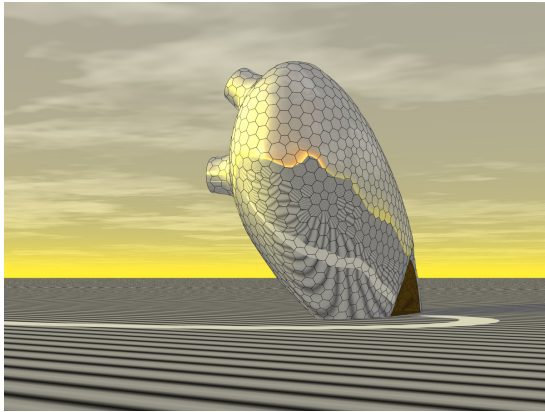
**Figure 4.12** – Effect of global vs spatially varying kink angle specifications on the Skipper Library dataset, along with statistics for corresponding paneling solutions (see also Figure 4.11).



(a) *Paneling solution with kink angle thresholds specified globally over the surface.*

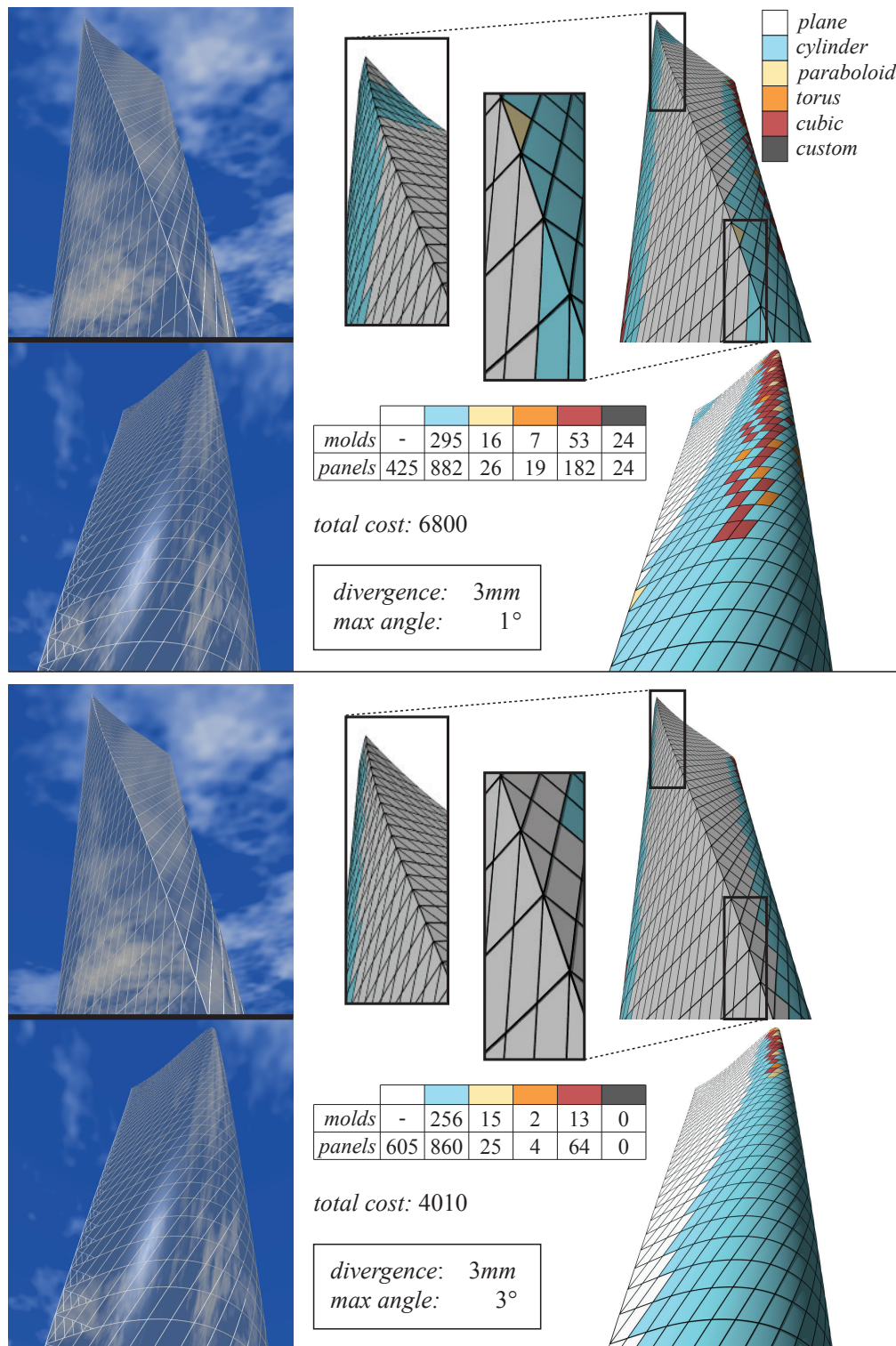


(b) *Paneling solution with spatially adaptive kink angle thresholds.*



(c) *Paneling solution with another set of spatially adaptive kink angle thresholds.*

**Figure 4.13** – *Effect of global vs spatially varying kink angle specifications on the Skipper Library dataset. Please refer to Figure 4.11 for details.*

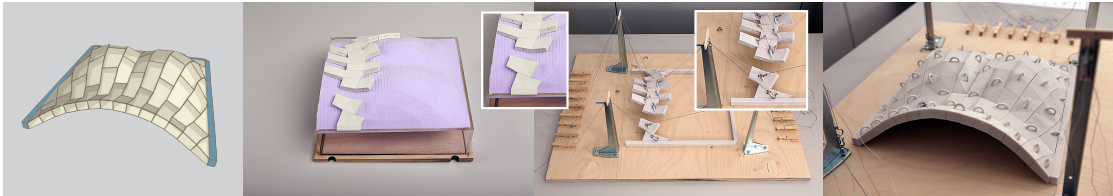


**Figure 4.14** – Paneling solution respecting crease line(s) on the input model. The characteristic sharp feature line of the Lissajous Tower is preserved in our paneling solution.





## 5 Assembling Self-Supporting Structures



**Figure 5.1** – We propose a construction method for self-supporting structures that uses chains, instead of a dense formwork, to support the blocks during the intermediate construction stages. Our algorithm finds a work-minimizing sequence that guides the construction of the structure, indicating which chains are necessary to guarantee stability at each step. From left to right: a self-supporting structure, an intermediate construction stage with dense formwork, an intermediate construction stage with our method and the assembled model.

Self-supporting structures are prominent in historical and contemporary architecture due to advantageous structural properties and efficient use of material. Computer graphics research has recently contributed new design tools that allow creating and interactively exploring self-supporting *freeform* designs. However, the physical construction of such freeform structures remains challenging, even on small scales. Current construction processes require extensive formwork during assembly, which quickly leads to prohibitively high construction costs for realizations on a building scale. This greatly limits the practical impact of the existing freeform design tools. We propose to replace the commonly used dense formwork with a sparse set of temporary chains. Our method enables gradual construction of the masonry model in stable sections and drastically reduces the material requirements and construction costs. We analyze the input using a variational method to find stable sections, and devise a computationally tractable divide-and-conquer strategy for the combinatorial problem of finding an optimal construction sequence. We validate our method on 3D printed models, demonstrate an application to the restoration of historical models, and create designs of recreational, collaborative self-supporting puzzles.

### 5.1 Introduction

The majority of man-made objects are composed of multiple inter-locking parts, kept together by glue, bolts or other connections. The division into components is often necessary to achieve a certain purpose (computers, cars) or to make the fabrication of large models feasible or cheaper (buildings, furniture, roads, railways, large 3D printed models, etc.).

In this work, we focus on the construction of self-supporting structures that are composed of bricks or stone blocks without any mortar to bind them together. Most of the world’s architectural heritage consist of self-supporting masonry structures that require no supporting framework, since the entire structure is in a static equilibrium configuration.

The design of modern, freeform self-supporting structures has recently received a lot of interest in computer graphics [VHWP12, LPS<sup>+</sup>13, dGAOD13, PBSH13], but their physical construction has only been addressed for small-scale models. The method proposed in [PBSH13] relies on *dense* formwork (Figure 5.1) to support all the blocks until the entire construction is completed; after the last piece is put in place, the structure is in equilibrium and the formwork can be carefully removed. This method is difficult to apply to large-scale structures, because a dense formwork able to sustain the weight of large stone blocks is too expensive and not practical, especially considering that the formwork has to be dismantled after all the blocks are in place. Also, removing the formwork demands technically complex and expensive solutions: the formwork has to be lowered evenly to avoid failures due to redistribution of forces. Due to the lack of an economically feasible construction strategy, freeform masonry structures are currently rarely built, despite their advantageous structural properties and unique aesthetics. Additionally, the majority of the cost is associated with the foundations necessary to support the formwork.

We propose a different approach, replacing the dense formwork with a sparse set of chains that are connected to fixed anchor points. While chains have been used for construction before, our method specifically aims at finding a work-minimizing assembly sequence, requiring as few chains to be rehung as possible. Our solution leverages the internal force distribution of the partially assembled structure and only provides the minimally required additional supports to keep the structure in static equilibrium at all stages of the assembly. The use of chains has been pioneered in modern construction by [Dre13], who successfully built simple self-supporting structures using one or more chains per block (Figure 5.2). Historical methods applied rope supports in arch construction [Fit61]. Our work extends this idea to general freeform self-supporting structures. We show that designs created with the methods of [VHWP12, LPS<sup>+</sup>13, dGAOD13, PBSH13] can all be handled by our algorithm.

The core problem that we address is finding a sequence of block insertions that minimizes a specific cost function defined on the number of chains required during assembly. This



**Figure 5.2** – *The Arch-Lock system [Dre13] is used to construct a simple arch (left), and a barrel vault (right). [Copyright photographs: Lock-Block Ltd. 2013]*

search problem is hard, because the space of assembly sequences is exponential in the number of blocks and anchor points. The situation is exacerbated by the fact that even verifying the force equilibrium of a single construction state is already computationally involved (see Section 5.2). As illustrated in Figure 5.3, naive solutions lead to an impractically high number of chains and are often not able to complete the structure since it is impossible to find valid configurations of chains that keep the structure in equilibrium for each state (Figures 5.3, 5.13, 5.14, and 5.15 are the only ones that can be constructed using a trivial z-filling sequence according to the equilibrium model we use). In feasible sequences, the z-ordering approximately doubles the number of times a chain needs to be rehung compared to our solution, which is significant for real construction.

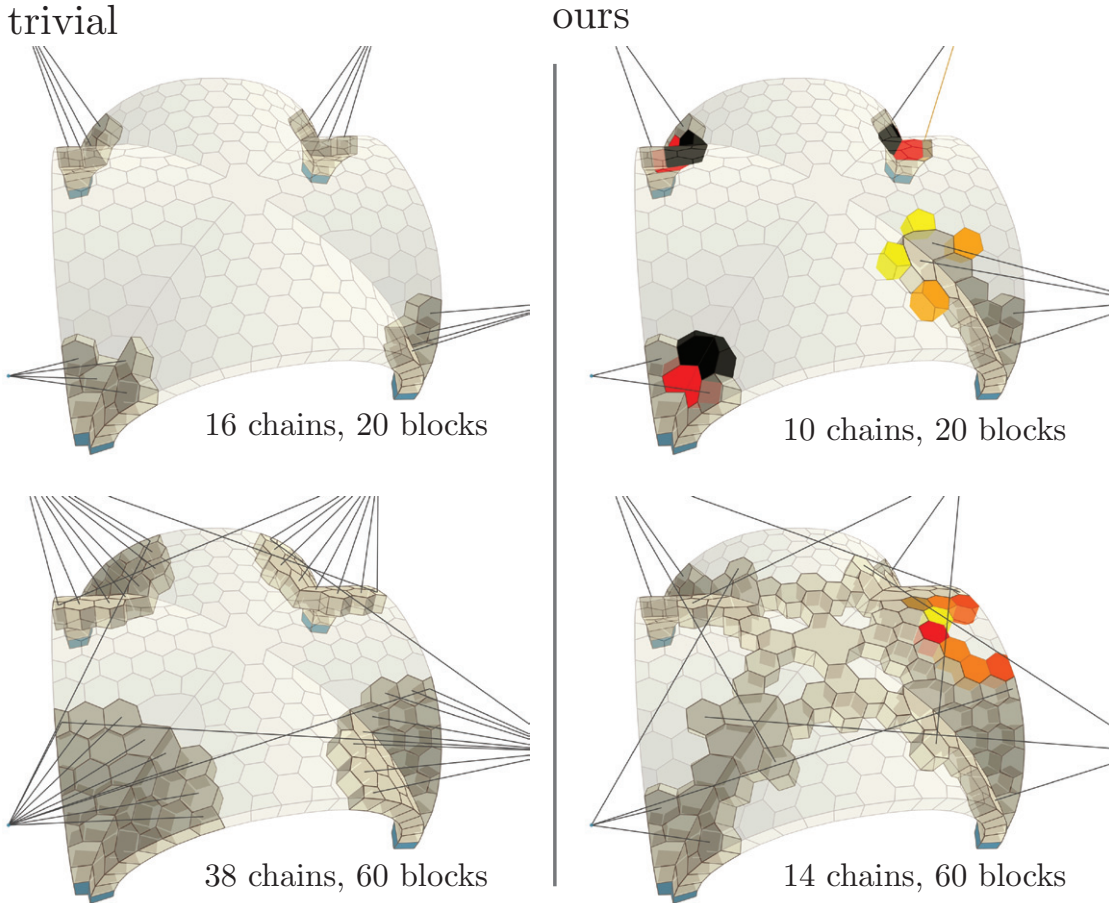
To make this combinatorial optimization problem computationally tractable, we introduce a divide-and-conquer strategy that first decomposes the design model into *stable sections*. This decomposition is computed using an optimization approach that applies sparsity-inducing norms to minimize the number of non-zero forces acting between blocks. Given the decomposition, we apply a greedy optimization to find the assembly sequence of the individual sections. While this strategy is not guaranteed to find the globally optimal solution, it greatly reduces the amount of work and chains compared to non-optimized construction sequences.

While our focus is on the fabrication of self-supporting surfaces, our contributions can be applied to other domains, such as restoration of existing structures and design of *self-supporting puzzles*. These can be printed with a consumer 3D printer and assembled by multiple players using fingers instead of chains.

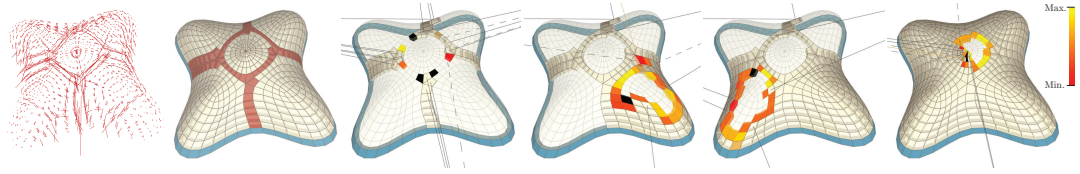
The contributions of this chapter are as follows:

1. We propose an alternative way of constructing masonry structures that requires a negligible amount of formwork compared to traditional techniques.
2. We present an optimization algorithm to analyze the equilibrium of a structure exposing its arches and implying a segmentation into stable sections. We propose an algorithm that leverages sparsity to minimize the number of chains that are necessary to avoid failures in the intermediate construction stages.
3. We validate our algorithm on physical examples of a small-scale 3D printed model and a self-supporting puzzle.

Work related to our method is discussed in Section 2.2



**Figure 5.3** – Two intermediate construction stages of our optimized sequence (right) and a trivial z-ordering (left). Our sequence needs considerably less work (0.62 instead of 1.13 chain changes per state in average), while computing it takes 3.5 times longer than determining the sparse set of chains for the trivial sequence.



**Figure 5.4** – Our algorithm converts an input masonry model in a work-minimizing construction sequence. From left to right: Forces resulting from our global equilibrium analysis, arch-blocks as extracted from flood-fill, four different states of the construction sequence. In all our figures, the blocks and chains are color-coded as follows: blue for support, light yellow for free blocks, gold for the newly added block and chains, dashed lines for chains that can be removed, and black lines for other active chains. We also color-code the candidate blocks considered by our sequence optimization using the minimum of Equation (5.5) relative to the other candidates in the color-bar on the right. Candidate blocks leading to an invalid state are shown in black.

## 5.2 Method

Figure 5.4 provides an overview of our algorithm to generate a work-minimizing construction sequence for a given self-supporting structure. After introducing some terminology (Section 5.2), we discuss how we verify static equilibrium in a collection of blocks and chains (Section 5.2.1). This method is an important component of our two-stage optimization algorithm. The first stage analyzes the equilibrium of the surface to find a set of stable regions (Section 5.2.3). Based on this decomposition, we generate a construction sequence using a greedy algorithm that initially constructs the arches separating the regions, and then fills the stable regions one by one (Section 5.2.4). We evaluate our algorithm in different scales and applications (see Section 5.3 for more details).

### Preliminaries

We model a masonry structure as a set of rigid blocks that are represented by closed manifold triangle meshes. The construction site is specified as a collection of *supports* and *anchor points* (see Figure 5.5). Each block can be in contact with other blocks or any of the supports. We call the contact surfaces between blocks *interfaces*. A block can also be attached to an anchor using a *chain*, represented as an inextensible straight line segment connecting the block and the anchor. Each block has a *hook* that provides an attachment point for the chains. Multiple chains can be attached to the same hook. In our examples, the block’s hook is placed at the intersection of the top face and the ray emanating from the center of gravity parallel to the direction of the normal of the top face. An intermediate *construction state* is a spatial arrangement of blocks and chains. We say that a state is *valid* if and only if it is in static equilibrium (Section 5.2.1).

Our algorithm converts an unordered collection of blocks into an *ordered construction sequence*, composed of valid construction states, each adding one single block to the

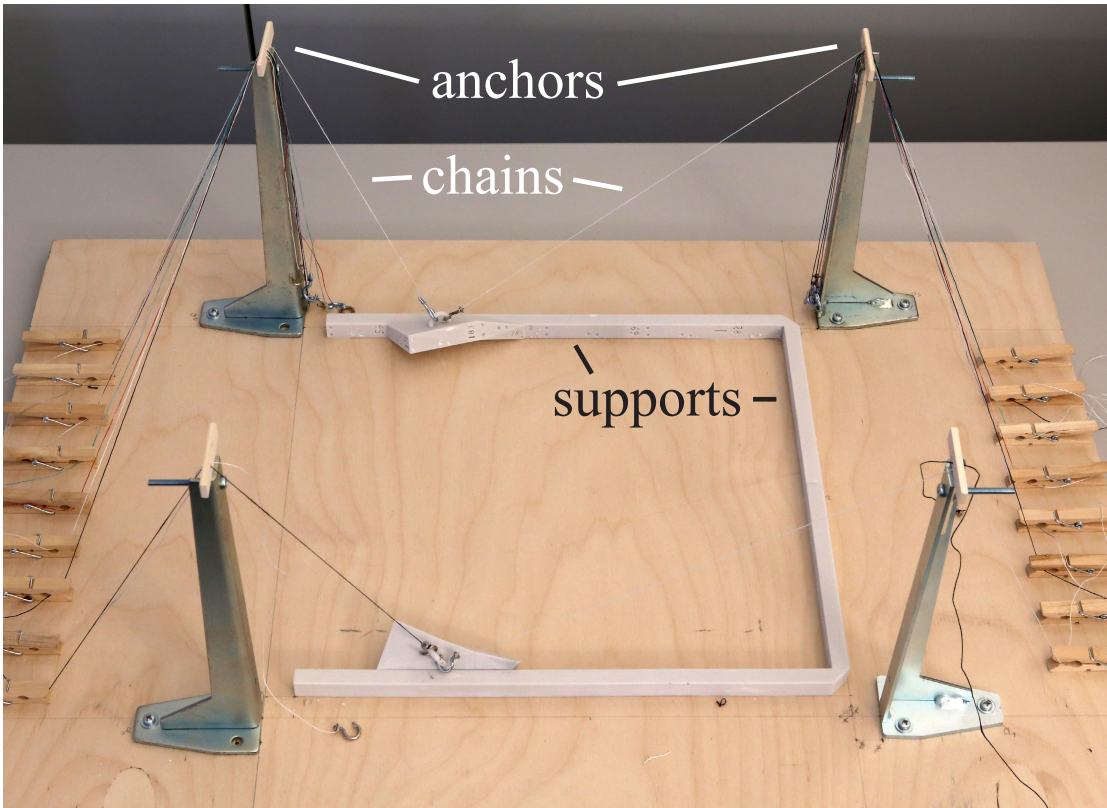


structure. Of the many construction sequences that exist for a given set of blocks and anchor positions, we strive to find one that minimizes the amount of *work* required in the construction.

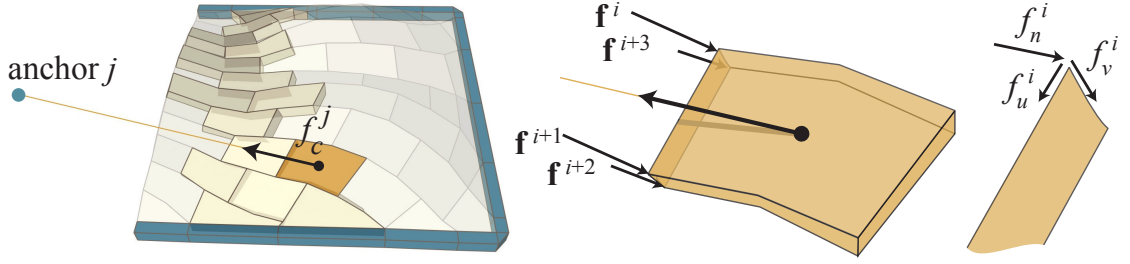
We define the work required to assemble a structure as the total number of chains added and removed during the construction. This definition is motivated by practical construction concerns: adding or removing a chain is an expensive operation that requires a considerable amount of time and energy, and thus directly relates to the construction cost.

### 5.2.1 Construction State Validity

An important component of our system is an algorithm to check whether an intermediate construction state is valid, i.e., the blocks are in static equilibrium. Traditional physical simulation methods are known to be unreliable in a masonry setting due to the extreme stiffness of the system [DeJ09]. We thus rely on the alternative model proposed in [WOD09], which we review in the following paragraphs. In this method, a state is valid if there exists a force distribution that satisfies a set of constraints. In general, those constraints admit more than one single distribution and do not fully determine the actual



**Figure 5.5** – *Construction site mockup.*



**Figure 5.6** – Model of chain forces and contact forces at interfaces between blocks.

forces acting. We leverage this indeterminacy by picking sparse force distributions to reveal a sparse valid subset of chains on one hand, and an approximately valid subset of blocks on the other hand.

We decompose each force  $\mathbf{f}^i$  acting on an interface between two blocks into its axial component  $f_n^i$ , perpendicular to the face, and two orthogonal in-plane friction components,  $f_u^i$  and  $f_v^i$  (Figure 5.6). We model one force vector per vertex of the contact-polygon between the two blocks. We encode the magnitude of the force introduced by the chain connecting a block  $\mathcal{B}$  to an anchor  $a$  with a scalar value  $f_c^{\mathcal{B},a}$  and its orientation by a unit vector  $\mathbf{d}^{\mathcal{B},a}$ . Note that we are interested in static equilibrium, so  $\mathbf{d}^{\mathcal{B},a}$  is fixed and only depends on the input geometry.

### 5.2.2 Static Equilibrium

Static equilibrium conditions require that net force and net torque acting on each block cancel out. In our model, we assume to have a small set of blocks anchored to the ground and a set of additional forces  $f_c$  acting on the blocks due to chain actions. Combining equilibrium constraints for each block yields a linear system of equations [Liv92]. For each block  $\mathcal{B}$ , the force distribution must satisfy the following equilibrium conditions:

$$\begin{bmatrix} [1.6] \sum_{i \in V(\mathcal{B})} \mathbf{f}^i + \sum_{a \in C(\mathcal{B})} f_c^{\mathcal{B},a} \mathbf{d}^{\mathcal{B},a} \\ \sum_{i \in V(\mathcal{B})} \mathcal{A}^{\mathcal{B},i} \mathbf{f}^i + \sum_{a \in C(\mathcal{B})} \mathcal{A}_c^{\mathcal{B},a} f_c^{\mathcal{B},a} \mathbf{d}^{\mathcal{B},a} \end{bmatrix} = \begin{bmatrix} [1.9] - \mathbf{g}_B \\ \mathbf{0} \end{bmatrix} \quad (5.1)$$

The top row corresponds to force equilibrium, where  $\mathbf{g}_B$  is a vector containing the gravity caused by block weight,  $V(\mathcal{B})$  is the set of all force indices acting on the interfaces of block  $\mathcal{B}$ , and  $C(\mathcal{B})$  is the set of all chains acting on block  $\mathcal{B}$ . The bottom row corresponds to torque equilibrium where matrix  $\mathcal{A}^{\mathcal{B},i}$  contains coefficients for the torque contribution of force  $\mathbf{f}^i$ , and coefficient matrix  $\mathcal{A}_c^{\mathcal{B},a}$  accounts for the torque contribution of the chain force  $f_c^{\mathcal{B},a}$  (see the Appendix in [WOD09]).

### Compression and Tension Constraints

According to the limit analysis of masonry, the material can be assumed to have zero tensile strength. This condition is expressed as a non-negativity constraint on the axial components of the forces:

$$f_n^i \geq 0. \quad (5.2)$$

Similarly, all chains can only introduce tensile forces:

$$f_c^j \leq 0. \quad (5.3)$$

### Friction Constraints

A friction constraint is applied at all block interfaces. For each triplet of forces  $\{f_n^i, f_u^i, f_v^i\}$ , the two in-plane forces are constrained within the friction cone of the normal force  $f_n$ . We linearize the friction constraints with a pyramid approximation:

$$|f_u^i|, |f_v^i| \leq \frac{\alpha}{\sqrt{2}} f_n^i, \quad \forall i \in \text{interface vertices} \quad (5.4)$$

where  $\alpha$  is the coefficient of static friction. More details are given in Section 5.4.3.

#### 5.2.3 Global Equilibrium Analysis

Historical self-supporting structures are highly regular and explicitly composed of *primary arches* that were constructed before the rest of the structure and used as support while building the other parts. Following this construction strategy, the expensive support material is used only for a small fraction of the structure, which then acts as a support for the other parts.

The lack of regularity and symmetry in freeform designs makes the manual identification of arches challenging, even for experts. Furthermore, freeform designs often do not contain any *exact* arches, i.e. , they do not have any subset of blocks that is in static equilibrium without any external support. However, any masonry structure contains one or more of what we term *quasi-arches*: subsets of blocks that require only a small number of chains to stand. Finding quasi-arches is even harder than finding arches, since it is a global problem that requires an understanding of how forces distribute in the entire structure. We propose a segmentation algorithm that analyzes the distribution of forces in a masonry structure and automatically extracts quasi-arches.



## Combinatorial Problem

Finding quasi-arches is equivalent to finding a maximal, non-trivial subset of blocks that can be removed from a masonry structure without collapsing the remaining part. Determining which blocks can be removed is a hard and ill-posed combinatorial problem, since we want to avoid the trivial solution which contains no blocks. A brute-force approach is infeasible, since testing for the validity of all possible construction sequences has exponential complexity. In the following section, we describe our proposed approach that makes the discovery of quasi-arches a tractable problem.

## Continuous Relaxation

Equations (5.1), (5.2), (5.3) and (5.4) define a convex constraint set containing all the possible internal force distributions for which the structure is in static equilibrium. By using a variational method, we can explore this constrained space using a sparsity inducing functional to find valid distributions that tend to concentrate the internal force on a small subset of interfaces and thus indicate potential quasi-arches in the model.

To convert the combinatorial problem into a computationally tractable continuous problem, we model only the side effects of removing a block. Each removal implies that the forces on its interfaces should be zero. We continuously relax this condition using a block-sparsity approach that searches for an equilibrium solution that can be explained with the majority of the forces concentrated on a small subset of the interfaces. Note that the trivial solution does not satisfy the equilibrium constraints (5.1), since they prohibit all interfaces and chains of a block to be zero.

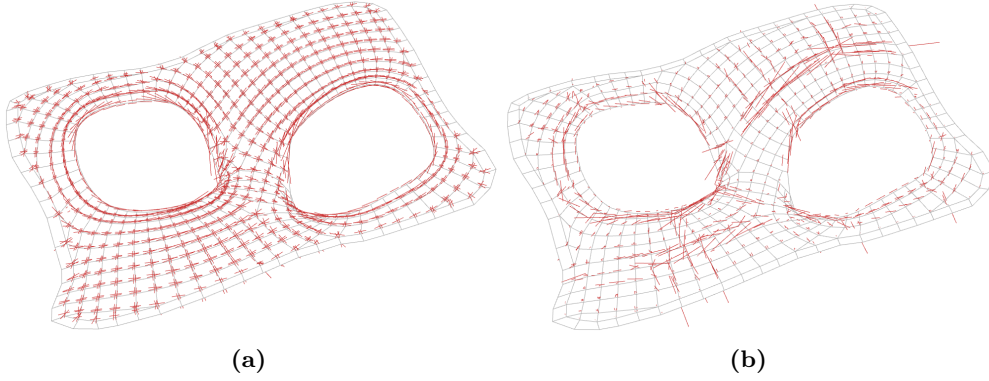
To induce sparsity on the interface forces, we first introduce a new scalar variable  $s_{\mathcal{I}}$  for each interface  $\mathcal{I}$  that bounds the magnitude of the resultant of the forces acting on it. We minimize the number of non-zero  $s_{\mathcal{I}}$  and  $f_c$  solving a  $L_p$ -relaxation (Section 5.4.1) of the following optimization problem with an iterative reweighting scheme (Section 5.4.2) :

$$\min_{f_u f_v f_n f_c} (1 - \lambda) \sum_{\mathcal{B}} \sum_a \mathbb{I}_0(f_c^{B,a}) + \lambda \sum_{\mathcal{I}} \mathbb{I}_0(s_{\mathcal{I}}) \quad (5.5)$$

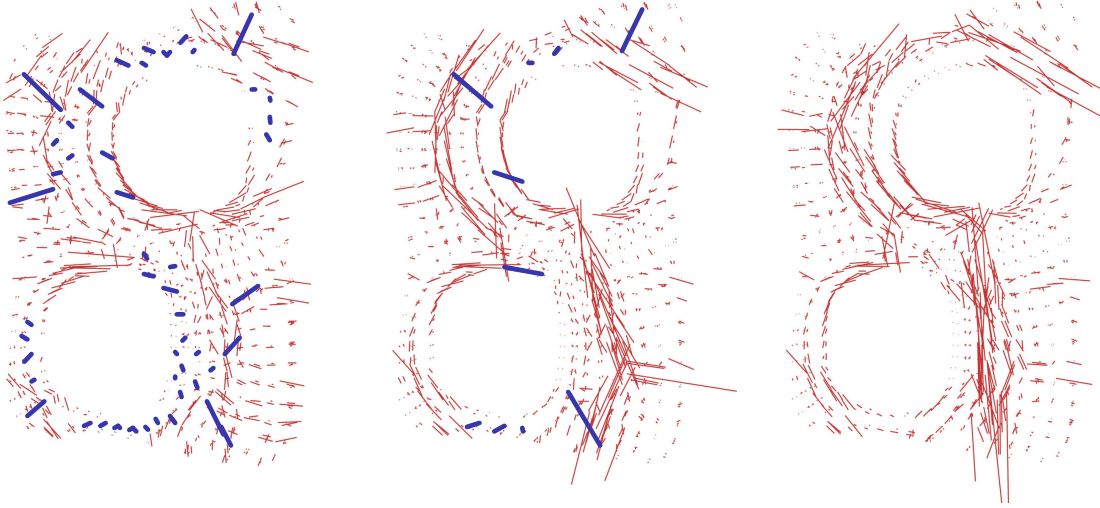
$$\text{s.t. } (5.1), (5.2), (5.3), (5.4), \quad (5.6)$$

$$s_{\mathcal{I}} \geq \left( \sum_{i \in \mathcal{I}} \|\mathbf{f}^i\|^2 \right)^{1/2} \quad (5.7)$$

where  $\mathbb{I}_0(x) = 1$  for  $x \neq 0$ ,  $\mathbb{I}_0(x) = 0$  otherwise. We use  $\lambda = 0.06$  in all our experiments. The functional contains two sparsity-enforcing terms: the first minimizes the number of introduced chains, while the second minimizes the number of used interfaces, see Figure 5.7. The minimization of the auxiliary variables  $s_{\mathcal{I}}$  together with the bound from below imply equality for Equation (5.7). At a minimum,  $s_{\mathcal{I}}$  therefore is equal to right-hand side, which is the resultant of the forces per interfaces. Note that the



**Figure 5.7** – Global analysis on a synthetic example. Force vectors (red lines) indicate the magnitude of forces between blocks.



**Figure 5.8** – The parameter  $\lambda$  controls the tradeoff between stable regions and introduced chains. Chain forces are shown as blue lines. From left to right,  $\lambda = 0.15, 0.12, 0.06$ .

equilibrium constraints are guaranteed to be satisfied after the minimization since they are enforced as hard constraints. The tradeoff between the number of stable regions and chains used is controlled by a single parameter  $\lambda \in [0, 1]$  (Figure 5.8). For each input model, we normalize the variables by choosing the material density such that the average length of the blocks' gravity forces is one. Our constraints are scale-independent with respect to density.

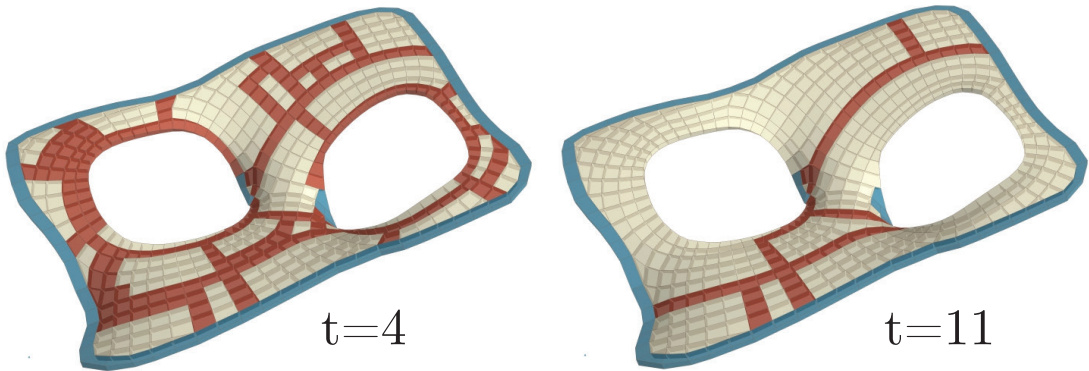
### Quasi-arch Extraction

Despite the sparsification, the weight of each block must still be redirected through the assembly down to the boundary or up along the chains to satisfy equilibrium equations. This naturally leads to a concentration of forces along linear subsets of blocks, even if the

minimization is not explicitly trying to concentrate the force distribution in connected components.

We extract the quasi-arches from the force distribution using a flood-fill approach restricted to grow across interfaces whose maximum of the normal component of its forces is above an user-defined threshold  $t$  (Figure 5.9). A quasi-arch is a connected component found by the flood-fill procedure that connects two supported blocks. The following steps extract connected arches and erode dense regions after the flood-fill: We reject a quasi-arch if it contains less than two support blocks, or if the bounding box diagonal of the centroids of support blocks is more than 10 times smaller than that of the whole model including anchor points. We then remove singly-connected blocks which are not supported. After minimizing Equation (5.5), the extraction can be computed in real-time, allowing the user to manually choose a value for  $t$  based on visual feedback.

**Remark.** The masonry crack prediction of Fraternali [Fra10] bears some similarity with our quasi-arch extraction: The method applies an iterative scheme to jointly find a discrete stress surface and a compressive stress distribution explaining its equilibrium via a thrust network approach, then predicts cracks in regions of zero or uniaxial compressive stress. In contrast, our quasi-arch extraction explicitly optimizes for sparse forces over all valid equilibrium solutions.



**Figure 5.9** – *Extracting quasi-arches.* This figure shows the quasi-arches extracted with two thresholds on the maximal normal component of interface forces.

### 5.2.4 Construction Sequence

We want to find a work-minimizing sequence to assemble each part of the structure using the global equilibrium analysis. To reduce the complexity of this combinatorial problem, we use a greedy approach that inserts a block at a time, taking local decisions. The work required for a certain step depends on the previous and successive states and it is thus impossible to minimize locally. Observing that the maximal work is bounded from above by the number of chains used, we opt for a strategy that directly minimizes the chains used, indirectly minimizing the work.

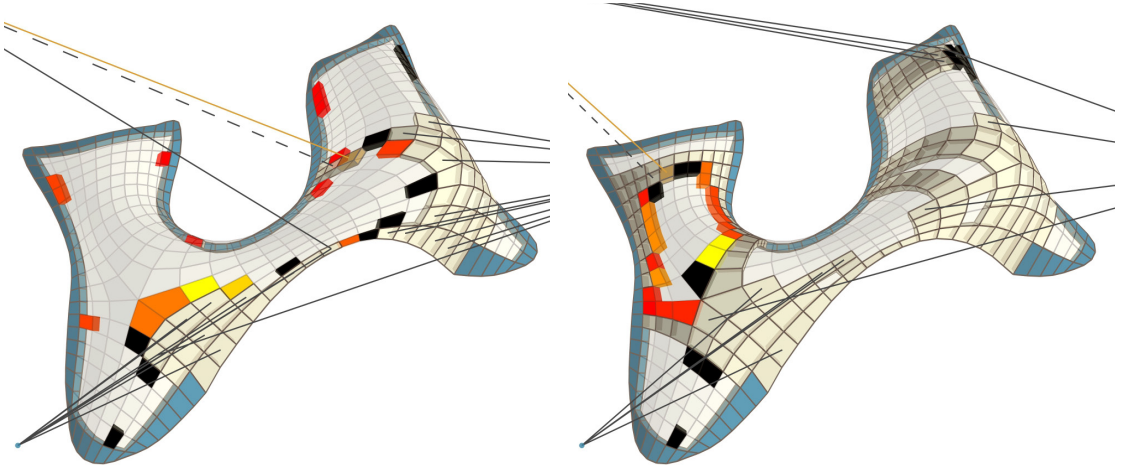
The detected quasi-arches are constructed first by restricting the greedy approach to only consider the blocks of the quasi-arches. After this stage, each remaining stable region is constructed using the same strategy, one at a time, starting with the region containing the most blocks. If the greedy approach fails to find a valid next block to insert, the next region is added to the candidate blocks. This heuristic drastically reduces the search space and consequently the computation time (5 times faster on Figure 5.3) and can reduce the work of the resulting sequence.

#### Next Block Insertion

Our algorithm attempts to independently insert each block that is connected to a support part or to any other block. For every inserted block, we solve the minimization problem in Equations (5.5) and (5.6), with  $\lambda = 0$ . If no equilibrium state can be found, we discard the configuration. We apply this procedure for all candidate blocks and select the valid configuration with the lowest cost. The cost is given by the  $L_p$ -relaxation of Equation (5.5), see Section 5.4.1. Directly using Equation (5.5) would potentially assign the same cost to many candidates. In Figure 5.10, we show the cost for inserting each of the candidate blocks in a specific intermediate configuration.

#### Optimization and Chain Pruning

To simplify the construction process, we only allow a chain to be introduced on newly inserted blocks, that is, whenever a chain is removed from a block, we do not allow our optimization to insert it again. In addition to simplifying construction, this heuristic also reduces the problem size. Note that we disable chain pruning for the comparisons with the trivial z-ordered sequences, since the pruning might prevent the z-ordering strategy to find a valid sequence.



**Figure 5.10** – At each construction step we test the work cost for inserting an additional block and we select the one with minimal energy. We highlight in black that blocks for which the optimization failed to find a force distribution that satisfies the equilibrium constraints.

### Trivial Filling Strategy Failure

We demonstrate on many examples that our approach is efficient and greatly reduces the work required to assemble the sequence. We show in Figure 5.3 a comparison between our approach and a height-ordered filling approach: our method requires around 50% less additions and removals of chains.

#### 5.2.5 Practical Constraints

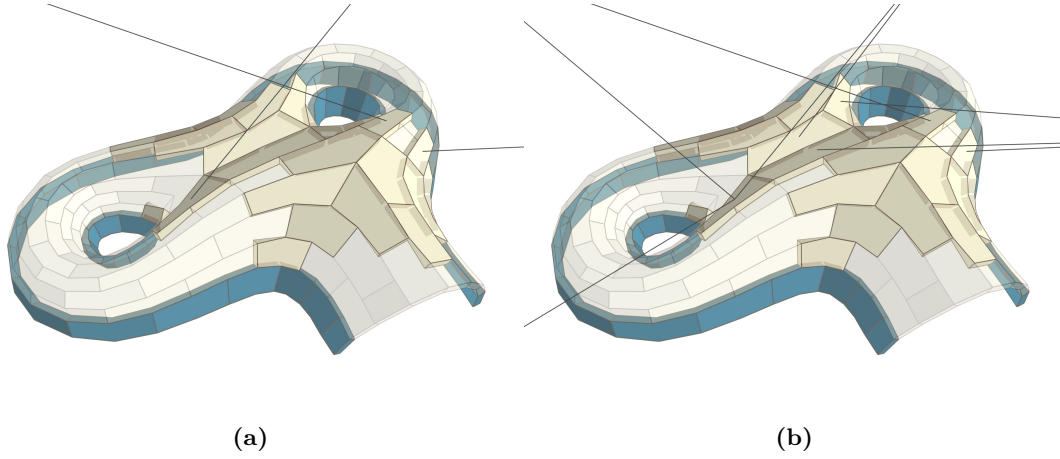
In practice, the anchors cannot support arbitrary forces; the chains will break if the tension is extreme. We therefore add constraints to our algorithm that account for bounds on chain forces. In addition, we introduce geometric and frictional safety factors to account for unavoidable fabrication inaccuracies as described below.

#### Anchor Bounds

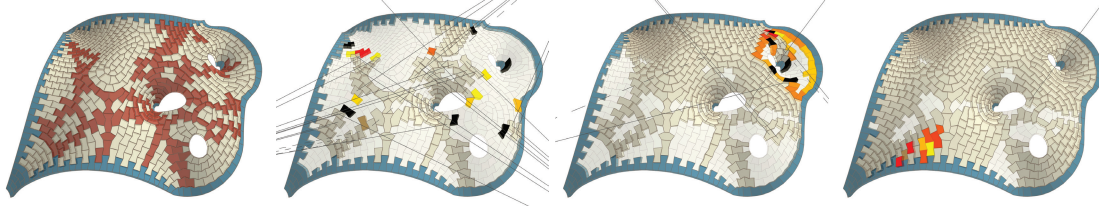
We conservatively bound the maximal force acting on an anchor  $a$  by adding a linear inequality constraint on the sum of the involved forces:

$$\sum_B f_c^{B,a} > -f^{max} \quad (5.8)$$

The condition is linear because we only sum the magnitude of the forces (all negative, see Equation 5.3), which are exactly the variables that we use in the optimization to represent the chain forces. Note that the bound is strict and our linearization is a



**Figure 5.11** – Adding a bound on the maximal force that an anchor can support generates a sequence that distributes the chain forces more evenly by adding additional chains. Left: Unbounded solution using 3 chains with max.  $f_c = 7.1$  and max. anchor bound 7.1. Note the max. chain is the only one connected to the max. anchor. Right: Solution with 8 chains bounded by 3.5 and anchors bounded by 5. For details please refer to Equations (5.8) and (5.9).



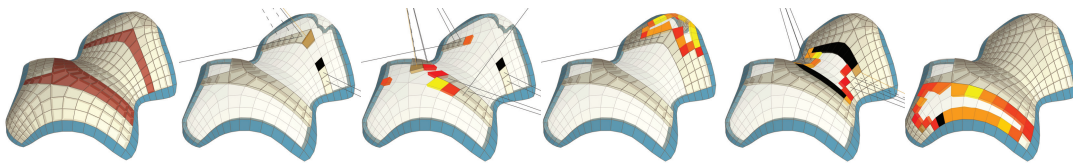
**Figure 5.12** – A large concert hall designed with RhinoVault is constructed using a sequence generated by our algorithm. The entire structure is made of hexagonal blocks placed using an interleaved layout typical of masonry constructions.

conservative estimate that does not take into account possible force cancellations. We show an example of a construction state with and without bounds on the anchors in Figure 5.11.

### Chain Bounds

Restricting the chain forces is a simple box constraint on the corresponding variables  $f_c^{\mathcal{B},a}$ , which directly represents its magnitude:

$$f_c^{\mathcal{B},a} < f_c^{\max} \quad (5.9)$$



**Figure 5.13** – A freeform self-supporting structure designed with [VHWP12]. We tessellated the surface with quadrilateral elements, which have no interlocking and are thus prone to sliding failures, requiring a considerable number of supporting chains to be stable in all intermediate stages.

### 5.2.6 Manufacturing Tolerances and Safety Factors

To account for manufacturing imprecisions and to incorporate a safety factor in the construction we introduce a geometric tolerance, to prevent torque failure, and a friction tolerance to prevent sliding failures. The former is achieved by uniformly downscaling the interface around the average of its vertices [Hey95], effectively reducing the space of stable equilibrium configurations and forcing our algorithm to introduce more chains. The second tolerance is on the friction parameter, which directly reduces the feasible space by attenuating the magnitude of the friction forces. For all our experiments, we used a 10% geometric tolerance and we conservatively set the friction coefficient  $\alpha$  to 0.6, which is approximately 10% lower than the value we experimentally measured on our 3D printed model.

### Intersecting Chains

To prevent chains from intersecting blocks during the construction sequence, we optimize only over the chains that do not intersect with any of the blocks. This set of chains is determined in a preprocessing step.

## 5.3 Results

We tested our algorithm on self-supporting surfaces designed with [RLB12] (Figures 5.12, 5.14), [PBSH13] (Figures 5.1, 5.9), [VHWP12] (Figures 5.4, 5.13, 5.11), [LPS<sup>+</sup>13] (Figure 5.10) and [dGAOD13] (Figure 5.3). We provide full construction sequences for all our results as short movie clips in the supplementary material. All our experiments were performed on a quad-core Intel i7 processor using the multi-threaded conic solver in the MOSEK optimization library [Mos14]. Statistics on the datasets and computation times are provided in Table 5.1. The optimization requires from a few minutes to a couple of hours, depending on the number of blocks. The computation time of the global analysis, in our largest models below 15 seconds, is negligible w.r.t. the sequence optimization. This comes without surprise, since we only solve a slightly bigger conic problem once, while the sequence needs many of them.

Model	#B	#C	#A	$t$	Avg. #C	Avg. $\Delta C$	Time
Figure 1	61	194	4	8	1.4	1.8	2
Figure 3	237	612	4	3.1	9.5	1.2	20
Figure 4	580	1652	4	4.6	4.9	1.2	249
Figure 8	203	1194	6	5	5.7	1.2	15
Figure 10	416	1127	4	0.9	9.4	1.1	95
Figure 11	128	306	4	0.6	2.0	0.5	9
Figure 12	588	2030	4	4.6	10.1	0.8	243
Figure 13	280	666	4	2.7	2.5	0.9	41
Figure 14	12	42	4	1	1.8	2.0	0.2

**Table 5.1** – Table with statistics for our results. From left to right: number of blocks, number of chains in the optimization, number of anchors, threshold parameter for the quasi-arch extraction, average number of used chains, average number of chain changes and computation time in minutes.

### 5.3.1 Large-scale Simulations

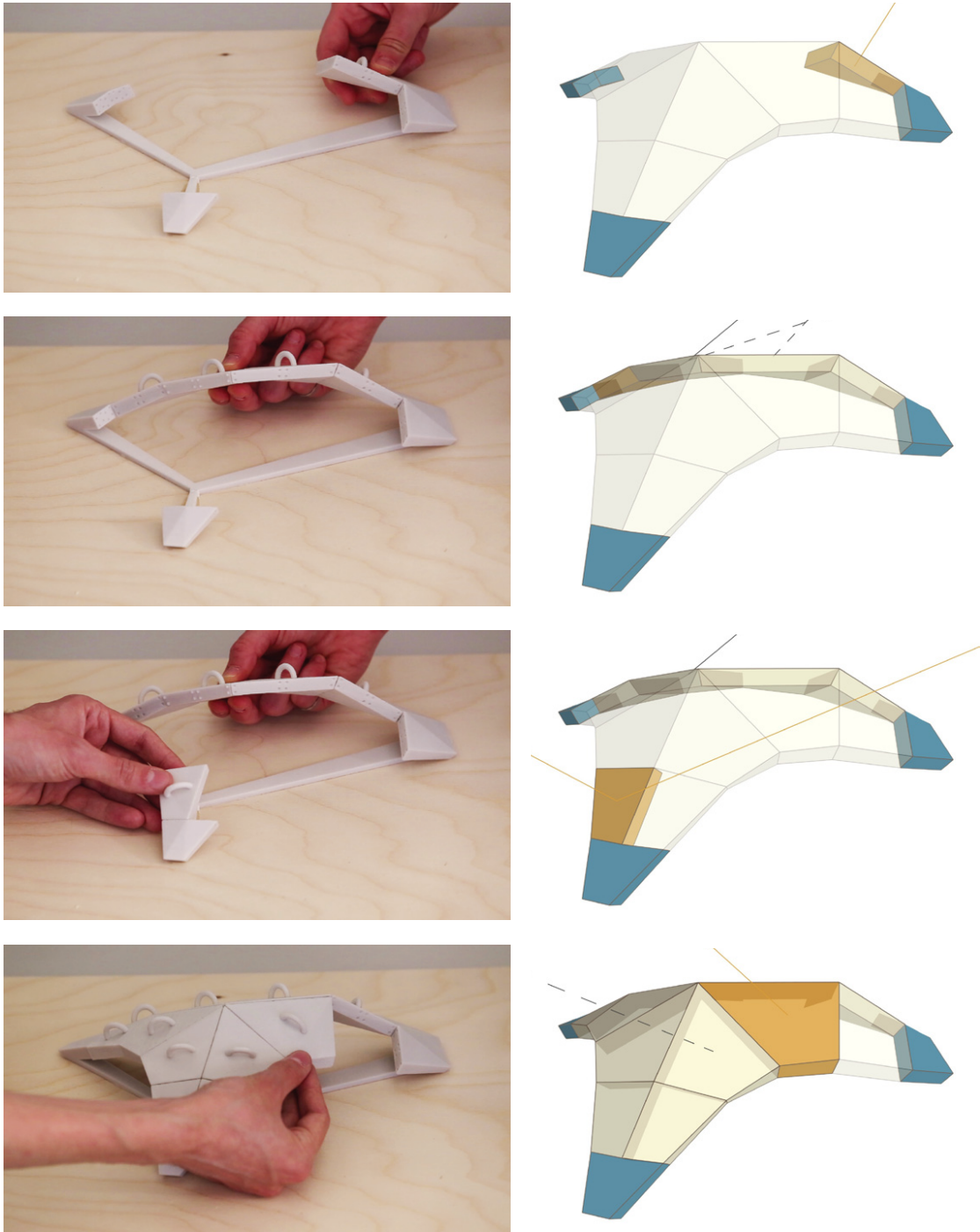
We test our algorithm on a complex self-supporting model designed using RhinoVault [RLB12] (Figure 5.12) and tessellated with a manually-designed staggered pattern.

To stress test our approach, we create construction sequences for two models tessellated with quads in Figure 5.4 and 5.13. These cases are particularly challenging because the lack of interlocking between the blocks makes these structures prone to friction failures.

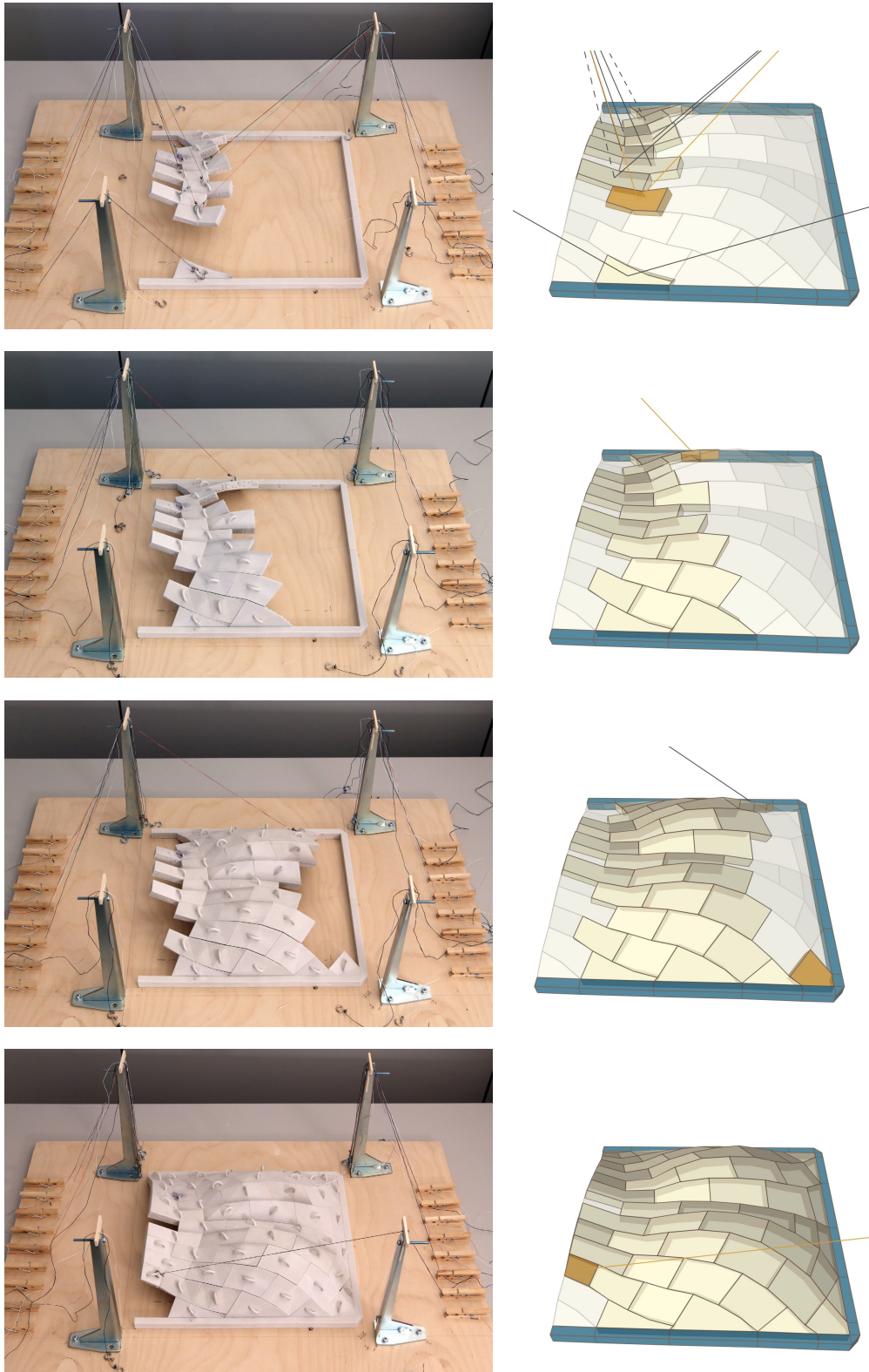
### 5.3.2 Self-supporting Puzzles

To demonstrate the flexibility of our algorithm, we designed the small self-supporting model in Figure 5.14 using RhinoVault [RLB12]. We used our algorithm to find a construction sequence with an upper bound of three blocks supported by chains at any given time. The conditions have been enforced by rejecting all the construction states where more than three blocks were connected to anchors. The construction sequence is the solution to the puzzle, which allows to build it with four hands: three to simulate the chain forces and one to insert the pieces. Our algorithm opens interesting possibilities to design complex 3D self-supporting collaborative puzzles, which we plan to explore in future works.





**Figure 5.14** – *Our algorithm can be used to design challenging physical puzzles.*



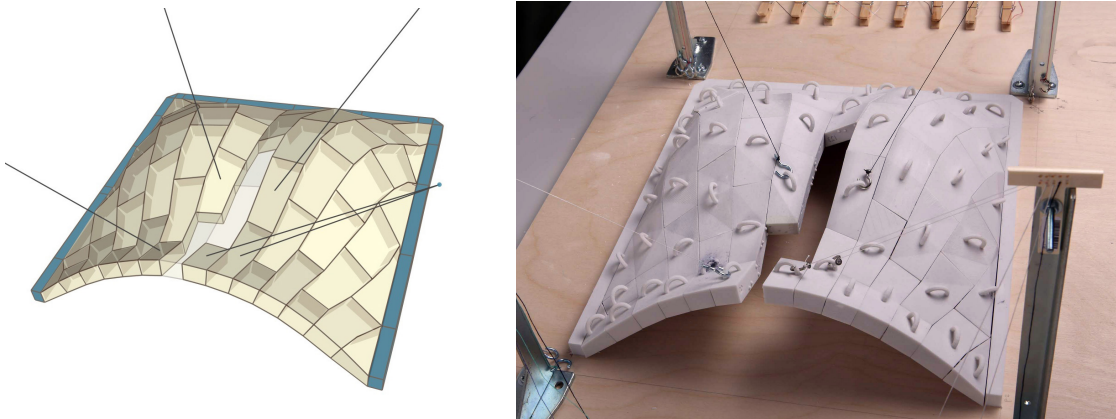
**Figure 5.15** – We validate our algorithm by constructing a masonry structure using our optimized work-minimizing construction sequence.

### 5.3.3 Validation via Small-scale Models

Since masonry is a problem of stability rather than stresses [Hey95], scaled block models can actually be used as structural models [ZLO10, VMMD12]. We validate our algorithm by 3D printing the blocks of a masonry structure designed with [PSSH13] and using metal hooks and sewing string to model the chains. We used our algorithm to generate the construction sequence and then physically constructed the model following all steps (Figure 5.15). We provide in the additional material the full sequence of photographs for each construction step, validating our simulation results. During the construction, we observed that the chains predicted by our algorithm are always in tension, suggesting that the equilibrium model we use accurately predicts the forces acting on the structure and does not introduce redundant chains.

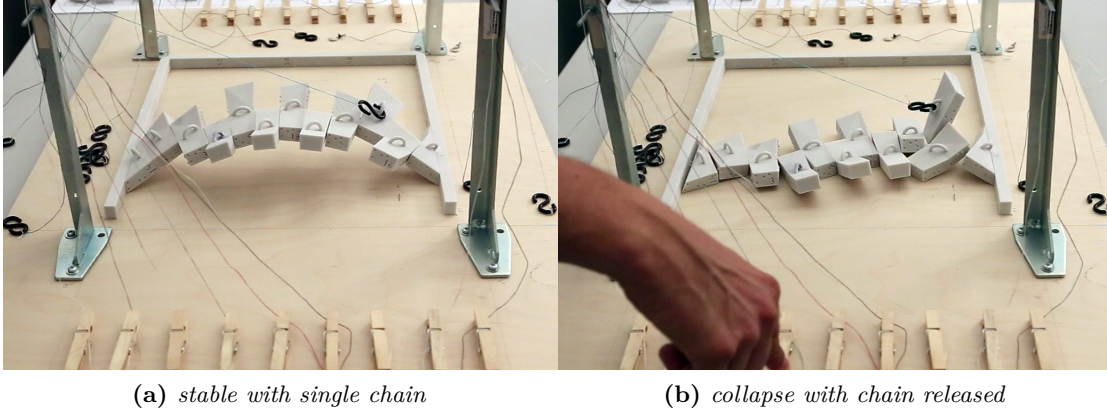
### 5.3.4 Restoration of Historical Buildings

Restoring an existing masonry building is a difficult task, since it is not possible to remove and replace blocks without risking a structural failure. Our method can be used to tackle this difficult problem, as shown in Figure 5.16. We optimize for a new equilibrium solution that does not contain the blocks we want to replace. The optimization redistributes the internal forces, and decides which chains should be inserted to guarantee the stability of the structure. After the chains are inserted, the blocks can be safely removed and restored. Note that intrusive techniques may be required such as drilling holes in the stones or gluing lightweight hooks/anchors.



**Figure 5.16** – *Our algorithm finds a sparse set of chains that guarantee stability even after the removal of a subset of the blocks. This can be applied to restoration of masonry structures.*





**Figure 5.17** – *This quasi-arch needs a chain to be stable. When the chain is loosened, the arch collapses due to a torque failure.*

### 5.3.5 Evaluation of Safety Factors

We designed an experiment to evaluate the accuracy of our safety factors and the equilibrium model we use. The global analysis step finds a quasi-arch in the middle of the structure only if we do not introduce a safety factor (Figure 5.17). Adding a friction and safety factor of 1% is sufficient to make the quasi-arch unstable and forcing our algorithm to introduce a chain. We reproduced this case with our 3D printed model, and verified that the arch is indeed extremely close to be in equilibrium, but cannot stand safely without a chain due to a torque failure.

## 5.4 Implementation

### 5.4.1 $L_p$ -Relaxation

Minimizing Equation (5.5) is a hard, discrete problem. We therefore use a continuous  $L_p$ -relaxation ( $0 < p < 1$ ) to approximate the minimization by

$$\min_{x_i} \sum_i w_i (r_i |x_i|^p) \quad \text{s.t. (5.6), (5.7),} \quad (5.10)$$

where  $r_i$  are additional, dynamic weights used for iterative reweighting as describe in the next section.  $w_i$  are weights of the discrete problem, e.g.  $1 - \lambda$  and  $\lambda$  in Equation (5.5). We experimentally found that  $p = 0.05$  produces the least work, see Table 5.2.

### 5.4.2 Iterative Reweighting

Iterative reweighting strategies convert non-linear  $L_p$ -minimization problems into a series of linear or quadratic problems. We choose an *iterative reweighted  $L_1$ -minimization*

(*IRL1*) [LLSZ13] to minimize Equation (5.10), in particular because each subproblem is a conic program that can be solved globally optimal. Suppose  $\hat{x}_i$  is a minimum of Equation (5.10) with  $r_i = 1$ . Then  $\hat{x}_i$  is also a minimum of the conic program in Equation (5.10) with  $p = 1$  and  $r_i = \hat{r}_i(\hat{x}_i)$ :

$$\hat{r}_i(x_i) = \frac{1}{|x_i|^{1-p} + \epsilon_r} \quad (5.11)$$

for  $\epsilon_r = 0$  and  $\hat{r}_i(0) = \infty$ . In practice we use  $\epsilon_r = 10^{-5}$  to avoid division by zero. However, we do not know  $\hat{x}_i$ . According to *IRL1* we iteratively estimate  $\hat{r}_i^t$  from the previous solution  $\hat{r}_i^t := \hat{r}_i(\hat{x}_i^{t-1})$ , leading to the following series of conic problems:

$$\hat{x}_i^t := \operatorname{argmin}_{x_i} \sum_i w_i(\hat{r}_i^t |x_i|) \quad \text{s.t. (5.6), (5.7),} \quad (5.12)$$

starting with  $\hat{r}_i^0 := 1$ . Note that the modulus can be replaced with the proper sign since all the variables subject to minimization are constrained to be either non-negative or non-positive according to Equation (5.2) and (5.3). We perform 5 iterations in our experiments. The resulting chain forces however are rarely precisely zero due to the relaxation and numerical precision. We therefore choose a parameter  $\epsilon_c = 10^{-8}$  below which we consider a chain to inactive. After minimizing Equation (5.12) we solve the first iteration again, this time setting variables below  $\epsilon_c$  to zero: In case we cannot find a solution, we reject the candidate block. Otherwise our construction sequence optimization picks the candidate minimizing Equation (5.10) with  $r_i = 1$ .

### 5.4.3 Friction Cones

Cone constraints of a conic program would be ideal to model the friction cone. In Equation (5.5), with  $\lambda > 0$ , some variables would then be part of multiple cone constraints, which is not allowed in the standard conic program formulation. We therefore use the conservative pyramidal approximation of Equation (5.4) for quasi-arch extraction, but use cone constraints for friction during the sequence optimization. The standard conic program does only allow cones of the form of Equation (5.7), but the friction cone needs a scalar

$p$	0.001	0.01	0.025	0.05	0.075	0.1	0.25	0.5	0.75	1
#C	7.36	6.51	3.51	1.28	1.16	1.31	2.05	2.84	8.03	18.33
$\Delta C$	1.41	1.34	1.25	1.02	1.15	1.21	1.31	1.41	1.41	3.25

**Table 5.2** – Average number of chains and chain changes for sequences of the model in Figure 5.1 computed with different parameter  $p$ . The sequences were computed without quasi-arches to isolate the influence of the parameter. Note that  $p = 1$  requires no reweighting.

coefficient  $s_K$  on the variable on the left-hand side. We therefore transform the conic program with scaled cones:

$$\begin{aligned} \min_x w^T x \quad & \text{s.t. } b_l \leq x \leq b_u, Ax \leq b, Cx = d, \\ \forall K \in \text{Cones} : \quad & s_K x_{K_0} \geq \left( \sum_i \|x_{K_i}\|^2 \right)^{1/2} \end{aligned}$$

where  $x_{K_0}$  is the left-hand side variable in cone  $K$ , and  $x_{K_i}$  summands of the right-hand side, into a standard conic program with the same minima:

$$\begin{aligned} \min_y (w^T D)^T y \quad & \text{s.t. } Db_l \leq y \leq Db_u, \\ (AD^{-1}) y \leq b, \quad & (CD^{-1}) y = d, \\ \forall K \in \text{Cones} : \quad & x_{K_0} \geq \left( \sum_i \|x_{K_i}\|^2 \right)^{1/2} \end{aligned}$$

where  $D$  is a diagonal matrix where  $D_{ii} = s_K$  if  $x_i$  appears in the left-hand side of cone  $K$ ,  $D_{ii} = 1$  otherwise. Note that this transformation is only meaningful if each variable only appears in the left-hand side of cones with the same  $s_K$ .

## 6 Discussion

This thesis presents methods of computational aid for realizing architectural structures, from modeling to cost-optimized fabrication and assembly. In this chapter, we first discuss each method individually, then compare how each of them handle constraints, and finally discuss directions of future work within the topic of the thesis.

### 6.1 ShapeOp

In Chapter 3, we presented ShapeOp, a robust and extensible geometric modeling paradigm. We explain the theoretical advantages over existing methods, and present the implementation as a simple, fast and extensible C++ library<sup>1</sup>. Our examples use the scripted grasshopper components provided with ShapeOp to highlight its practical importance.

ShapeOp is a continuous modeling tool directly incorporating various constraints in the modeling process. In the spirit of “what you see is what you get”, ShapeOp aims at restricting the user to only see models that satisfy the constraint and therefore qualifies as a constraint-aware modeling tool. If used for fabrication-constraints such as planarity of polygons, it acts as a fabrication-aware modeling tool.

Numerical optimizations such non-linear least-squares can be time-consuming, due to the need for evaluating the Jacobian and solving a different linear system in each iteration of the solver [NW06]. Therefore, although this approach works well for optimizing a single shape, it is not suitable for exploring the design space, where shapes need to be computed continuously in real time according to current input from the user. In comparison, each iteration of ShapeOp only involves parallel evaluation of projection operators, as well as the solution of a pre-factorized linear system. Such low computational cost makes ShapeOp a good choice for interactive constraint-based design. Recently, Tang

---

<sup>1</sup>[www.shapeop.org](http://www.shapeop.org)

et al [TSG<sup>+</sup>14] proposed a form-finding technique for polyhedral meshes, with much better performance than classical non-linear least squares formulations. However, their approach still relies on solving different linear systems in each iteration, resulting in poor performance for meshes with more than a few thousand vertices. On the contrary, the fixed linear system in ShapeOp makes it suitable even for large models.

Unlike force-based solvers, ShapeOp computes the equilibrium state of a system by minimizing a potential energy that incorporates physical forces as well as geometric constraints. Using the carefully designed numerical solver in ShapeOp, a stable solution can be computed in a small number of iterations with low computational cost, achieving better stability and efficiency than force-based solvers.

As future work, we plan to further explore the combination of continuous and discrete constraints. It is a common feature of many design problems to require some components to be selected from a finite set of choices, e.g. to allow for mass-production or adhere to standardized offerings. Unfortunately, such a finite set can be too restrictive to satisfy other criteria on the design. The optimization could potentially be enhanced to automatically detect a sparse set of constraints that needs to be violated in order to better preserve the design intent. This enhancement would have great potential to address challenges in constrained modeling.

## 6.2 Paneling

In Chapter 4, we presented a method for computing cost-optimized panelings. This paneling method tackles the difficult problem of choosing a set of molds, an assignment of a mold for each panel and a panel placement on the structure while minimizing cost and respecting tolerances in distance to the input surface and continuity.

Our paneling method presents improvements of the paneling algorithm introduced by Eigensatz and coworkers [EKS<sup>+</sup>10] to enable the preservation of sharp feature lines and the adaptive control of tolerance margins, allowing advanced exploration of cost effective rationalizations of architectural freeform surfaces. In our case studies on cutting edge architectural designs we evaluate the various modes of control enabled by our extended paneling algorithm and demonstrate the effectiveness of the algorithm with new examples, focusing on practical aspects complementary to the ones presented in [EKS<sup>+</sup>10].

The input to our paneling algorithm is a design surface and a set of curves (panelization seams) that define how the surface is divided into panels. We consider both the surface and the panelization seams as design intent and thus aim to change them as little as possible.



This approach leads to the following implications:

1. If the design surface or seams inherently violate the limits of a certain material or production process, for example with respect to maximum panel sizes, then the paneling algorithm will not eliminate this.
2. When computing minimum cost solutions, the paneling algorithm cares about cost of panel production only. This is reasonable because it just minimally changes the design surface and panelization seams, and therefore does not influence the cost of further parts like the substructure.

There are a few desirable extensions to the paneling algorithm leading to challenging problems for future research.

An obvious possibility for extending the paneling algorithm concerns the support of further mold types. We plan to include simple additional types like cones, but also more general surface types like general ruled surfaces. This would involve finding bounds to the new type of component to be supported for the set-cover approximation.

Figure 4.3 compares the paneling algorithm with rationalization approaches given by planar quad meshes and developable strip models. The latter include favorable geometric properties for the layout of substructures. It is natural to ask for possibilities of combining these approaches with the paneling algorithm. This motivates an adaption of the paneling algorithm towards the incorporation of optimization goals for the curve network, for example with respect to offsets and supporting structures.

For the three presented case studies—the Facade Design Study, the Lissajous Tower, and the Skipper Library—the paneling solutions are obtained in roughly 10 minutes, 1 hour, and 10 hours, respectively. In the future, we plan to explore both algorithmic and computational changes to speed up the process in order to ultimately enable interactive and simultaneous exploration of reference surface design (continuous modeling), curve network layout (combinatorial modeling), and paneling solutions. Such a method would then qualify as a paneling-aware modeling tool.

## 6.3 Assembly

In Chapter 5, we presented a method to assemble self-supporting structures using a sparse set of chains instead of a dense formwork. Our algorithm can process models generated with any of the existing design methods for self-supporting structures and can incorporate practical construction constraints in the optimization. Our method qualifies as a rationalization tool in the general sense of making an architectural structure realizable by reducing expenses.

We assume the construction site, and the masonry structure are given as input, and we focus on optimizing a valid construction sequence. An interesting venue for future work is a relaxation of this problem, where the algorithm is allowed to alter the anchor and hook placements. This would greatly increase the solution space and can further reduce the construction cost, but is extremely challenging since moving hooks and anchors has a global effect on all the construction states. Additionally, surface shape and/or tessellation could also be optimized.

Our greedy sequence optimization may fail when none of the candidate blocks can be inserted. We handled this case by manually perturbing the anchor positions to seek a possible solution. In particular, we moved the anchors to reduce the intersections between candidate chains and blocks, expanding the solution space. Relocating or adding anchor points automatically will be an interesting venue for future work. Alternatively, a backtracking strategy could be employed to further explore the space of valid sequences.

Explicitly maximizing temporal smoothness on the (in)active chains could theoretically help our method to reduce work. This maximization is hard to implement in our framework since the chain force magnitudes can vary substantially between states, while the corresponding chain stays active. A simple smoothness term on the magnitudes could therefore lead to denser solutions, producing sequences needing more work.

The static equilibrium analysis we use is based on the lower bound theorem [Hey95] which assumes hinging failure between blocks. While sliding failure is not accounted for, we guarantee existence of a feasible static equilibrium solution which satisfies the friction cone constraint. Further, we use a conservative coefficient of friction. Stability of masonry structures under sliding is an active area of research. Hinging is predominantly considered the limiting constraint in masonry analysis, particularly for arched and domed structures [Liv92].

More research will be required to address the additional constraints of large-scale constructions sites, but we did validate our algorithm on small scale, 3D printed models. We demonstrated that our algorithm can be used to design interesting physical puzzles, using hands instead of chains.

In our 3D printed models, we currently use small registration spheres (hemispheres added to the interface, respectively carved out from the interface in contact) to ensure that the construction is precise. They are not necessary but they greatly simplify construction since exact block alignment is difficult at a small scale. However, we found that small errors in chain length and fabrication technology (i.e. imbuing with glue powder-printed 3D blocks) can quickly sum up in an error of a few millimeters, making the construction difficult. We believe that a better locking mechanism between the blocks could be introduced to ameliorate this problem, or a computer vision system could be developed to help the exact placement of each block. Errors with fabrication tolerance would be

negligible at full scale. The chain stretching is proportional to the load and dependent on the cables' properties (cross-sectional area and axial stiffness), and can be computed precisely. In a real scenario, the cables should be adjusted after each construction step to compensate for the changing load. In our 3D printed models, we assumed chains do not stretch and adjusted the chain length manually when first introduced.

The quasi-arches have an unexplored advantage: They might allow using full, curvilinear formwork (from below) to fix only the quasi-arches, decoupling the equilibrium of each stable region. The stable regions could then be constructed in parallel, drastically speeding up the construction time.

It could be extended to structures which are not self-supporting, e.g. some that have explicit joints, by adapting the equilibrium model and redefining the cost function. Our method can also simply be used to determine a sparse set of chains or poles to support architectural structures assembled from components.

We believe that this work may have an impact both in digital fabrication, where the optimization over construction sequences is mostly unexplored, and in architecture. We demonstrated that it is possible to build self-supporting models with a negligible material overhead, and we hope that architects and engineers will be inspired by our work and apply this construction method to large-scale constructions.

The C++ source code of our implementation is available for download at <http://lgg.epfl.ch/selfassembly>.

## 6.4 Implicit and Explicit Constraints

It is interesting to observe how each of the methods presented in this thesis represents constraints differently. We can distinguish two ways of encoding constraints: *explicitly*, by computation, for example the planarity of a polygon in a mesh and *implicitly*, in the geometric representation, for example coinciding edges of neighboring faces in a polygon mesh. Implicit constraints are satisfied by construction.

In ShapeOp, projections are used to enforce constraints explicitly in an alternating minimization scheme. When a single point is involved in many explicit constraints, ShapeOp splits the point into one copy per constraint in the projection phase, but then enforces all copies to coincide in the global step by solving for a single point in the least-squares sense.

In contrast, our paneling method represents a curve network kept close to the input surface by a soft constraint, and a panel per face of the curve network, also linked by a soft closeness constraint. The panels themselves however are represented by a type (plane, cylinder, paraboloid, torus, cubic or custom) and the necessary parameter. This

enforces the panel shape as an implicit constraint.

In our assembly method, the blocks, anchor points and chains are fixed. We solve for an assembly sequence of the blocks and forces on the chains. The chains could be represented by boolean variables saying if they are needed, or not. The corresponding optimization, however, would be discrete and hard to approach. Our method relaxes the variables to be continuous, real force magnitudes, but then enforces a choice of active chains by means of a sparsifying optimization. This approach has been useful in many practical applications and it also allows to add an upper bound to the forces.

### 6.5 Future Work

The three methods presented in this thesis lay the groundwork for a sequential tool that handles modeling, mass-fabrication and assembly of architectural structures. While ShapeOp offers interactive response, the paneling and assembly methods can take up to several hours for a run. It is likely that the necessary speed-up for the latter two methods to ultimately become paneling- and assembly-aware modeling tools will only be achieved by a fundamentally different formalization and optimization approach. They are useful as is though because they only need a small fraction of the time necessary to actually build a full-scale structure. Also, ShapeOp is not capable of indicating changes in the combinatorics of a model as-is, but the user can rely on a combinatorial rationalization method for good initializations if available.

A modeling tool unifying all three aspects of architectural structures discussed here in an interactive process seems out of reach in the near future. And even if such a tool existed, there are still fundamental challenges in constrained combinatorial modeling, understanding of proper formalizations involving hard constraints and finding appropriate level of abstractions that need to be addressed. Also, approaches to more global understanding of general constrained design spaces are necessary for thorough exploration. This, however, presents itself as a very difficult problem which probably does not have a meaningful, simple solution.

This thesis approaches the general problems discussed in the last paragraph by studying specific instances and introducing meaningful assumptions. The resulting tools give some examples of what is to come and are also already useful in practice.

# Bibliography

- [AAS<sup>+</sup>09] Ramtin Attar, Robert Aish, Jos Stam, Duncan Brinsmead, Alex Tessier, Michael Glueck, and Azam Khan. Physics-based generative design. In *CAAD Futures Conference*, pages 231–244, 2009.
- [AM93] Sunil Arya and David M. Mount. Approximate nearest neighbor queries in fixed dimensions. In *SODA '93: Proceedings of the fourth annual ACM-SIAM Symposium on Discrete algorithms*, pages 271–280, Philadelphia, PA, USA, 1993. Society for Industrial and Applied Mathematics.
- [APH<sup>+</sup>03] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Trans. Graph.*, 22(3):828–837, 2003.
- [BBO<sup>+</sup>10] Bernd Bickel, Moritz Bächer, Miguel A. Otaduy, Hyunho Richard Lee, Hanspeter Pfister, Markus Gross, and Wojciech Matusik. Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graph.*, 29(4):63:1–63:10, 2010.
- [BDS<sup>+</sup>12] Sofien Bouaziz, Mario Deuss, Yuliy Schwartzburg, Thibaut Weise, and Mark Pauly. Shape-up: Shaping discrete geometry with projections. *Comp. Graph. Forum*, 31(5):1657–1667, August 2012.
- [BLP<sup>+</sup>13] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. *Computer Graphics Forum*, 32(6):51–76, 2013.
- [BM92] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14:239–256, 1992.
- [BML<sup>+</sup>14] Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4):154:1–154:11, 2014.
- [BPK<sup>+</sup>11] Pengbo Bo, Helmut Pottmann, Martin Kilian, Wenping Wang, and Jo-

## Bibliography

---

- hannes Wallner. Circular arc structures. *ACM Trans. Graph.*, 30(4):101:1–101:12, 2011.
- [BPW14] Michael Bartoň, Helmut Pottmann, and Johannes Wallner. Detection and reconstruction of freeform sweeps. *Computer Graphics Forum*, 33(2):23–32, 2014.
- [BS08] Alexander Bobenko and Yuri Suris. *Discrete differential geometry: Integrable Structure*. Number 98 in Graduate Studies in Math. American Math. Soc., 2008.
- [BZK09] David Bommes, Henrik Zimmer, and Leif Kobbelt. Mixed-integer quadrangulation. *ACM Trans. Graph.*, 28(3):77:1–77:10, July 2009.
- [Com94] P.L. Combettes. Inconsistent signal feasibility problems: Least-squares solutions in a product space. *IEEE Transactions on Signal Processing*, 42:2955–2966, 1994.
- [CPMS14] Paolo Cignoni, Nico Pietroni, Luigi Malomo, and Roberto Scopigno. Field-aligned mesh joinery. *ACM Trans. Graph.*, 33(1):11:1–11:12, 2014.
- [CSAD04] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. Variational shape approximation. *ACM Trans. Graph.*, 23(3):905–914, August 2004.
- [CTN<sup>+</sup>13] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Trans. Graph.*, 32(4):83:1–83:12, 2013.
- [Day65] A. S. Day. An introduction to dynamic relaxation. *The Engineer*, 219:218–221, 1965.
- [DBD<sup>+</sup>13] Bailin Deng, Sofien Bouaziz, Mario Deuss, Juyong Zhang, Yuliy Schwartzburg, and Mark Pauly. Exploring local modifications for constrained meshes. *Computer Graphics Forum*, 32(2pt1):11–20, 2013.
- [DBD<sup>+</sup>15] Bailin Deng, Sofien Bouaziz, Mario Deuss, Alexandre Kaspar, Yuliy Schwartzburg, and Mark Pauly. Interactive design exploration for constrained meshes. *Computer-Aided Design*, 61(0):13 – 23, 2015. Steering Architectural Form.
- [dC76] Manfredo do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976.
- [DDB<sup>+</sup>15] Mario Deuss, Anders Holden Deleuran, Sofien Bouaziz, Bailin Deng, Daniel

- Piker, and Mark Pauly. Shapeop - a robust and extensible geometric modeling paradigm. *Lectures of the Design Modelling Symposium*, 2015.
- [DeJ09] Matthew J. DeJong. *Seismic Assessment Strategies for Masonry Structures*. PhD thesis, MIT, 2009.
- [dGAOD13] Fernando de Goes, Pierre Alliez, Houman Owhadi, and Mathieu Desbrun. On the equilibrium of simplicial masonry structures. *ACM Trans. Graph.*, 32(4):93:1–93:10, 2013.
- [DPW11] Bailin Deng, Helmut Pottmann, and Johannes Wallner. Functional webs for freeform architecture. *Computer Graphics Forum*, 30(5):1369–1378, 2011.
- [DPW<sup>+</sup>14] Mario Deuss, Daniele Panozzo, Emily Whiting, Yang Liu, Philippe Block, Olga Sorkine-Hornung, and Mark Pauly. Assembling self-supporting structures. *ACM Trans. Graph.*, 33(6):214:1–214:10, November 2014.
- [Dre13] Jay Drew. United Lock-Block Ltd., 2013. <http://www.lockblock.com>.
- [DRPB12] Lara Davis, Matthias Rippmann, Tom Pawlofsky, and Philippe Block. Innovative funicular tile vaulting: A prototype in switzerland. *The Structural Engineer*, 90(11):46–56, 2012.
- [DVPSH14] Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. Designing  $n$ -PolyVector fields with complex polynomials. *Computer Graphics Forum (proceedings of EUROGRAPHICS Symposium on Geometry Processing)*, 33(5):1–11, 2014.
- [EDS<sup>+</sup>10] Michael Eigensatz, Mario Deuss, Alexander Schiftner, Martin Kilian, Niloy J. Mitra, Helmut Pottmann, and Mark Pauly. Case studies in cost-optimized paneling of architectural freeform surfaces. *Advances in Architectural Geometry 2010*, pages 49–72, 2010.
- [EKS<sup>+</sup>10] Michael Eigensatz, Martin Kilian, Alexander Schiftner, Niloy J. Mitra, Helmut Pottmann, and Mark Pauly. Paneling architectural freeform surfaces. *ACM Trans. Graph.*, 29(4):45:1–45:10, 2010.
- [Fal12] Giuseppe Fallacara. *Stereotomy: Stone Architecture and New Research*. Presses Ponts et Chaussées, 2012.
- [Fei98] Uriel Feige. A threshold of  $\ln n$  for approximating set cover. *JACM*, 45(4):634–652, 1998.
- [Fit61] John Fitchen. *The Construction of Gothic Cathedrals: A Study of Medieval*

- Vault Erection*. University of Chicago Press, 1961.
- [FLHCO10] Chi-Wing Fu, Chi-Fu Lai, Ying He, and Daniel Cohen-Or. K-set tilable surfaces. *ACM Trans. Graph.*, 29(4):44:1–44:6, 2010.
- [Fra10] Fernando Fraternali. A thrust network approach to the equilibrium problem of unreinforced masonry vaults via polyhedral stress functions. *Mechanics Research Communications*, 37(2):198 – 204, 2010.
- [GH95] Walter Gander and Jiri Hrebicek. *Solving Problems in Scientific Computing Using Maple and MATLAB*. Springer-Verlag New York, 1995.
- [GPF09] Aleksey Golovinskiy, Joshua Podolak, and Thomas Funkhouser. Symmetry-aware mesh processing. *Mathematics of Surfaces 2009 (invited paper)*, 2009. to appear.
- [GSC<sup>+</sup>02] James Glymph, Dennis Shelden, Cristiano Ceccato, Judith Mussel, and Hans Schober. A parametric strategy for freeform glass structures using quadrilateral planar facets. In *Acadia 2002*, pages 303–321. ACM, 2002.
- [HBA12] Kristian Hildebrand, Bernd Bickel, and Marc Alexa. Crdbrd: Shape fabrication by sliding planar slices. *Comput. Graph. Forum*, 31(2):583–592, 2012.
- [HD14] Michael Hansmeyer and Benjamin Dillenburger. Digital grotesque. <http://www.digital-grotesque.com>, 2014.
- [HEB15] Mathieu Huard, Michael Eigensatz, and Philippe Bompas. Planar panelization with extreme repetition. In Philippe Block, Jan Knippers, Niloy J. Mitra, and Wenping Wang, editors, *Advances in Architectural Geometry 2014*, pages 259–279. Springer International Publishing, 2015.
- [Hey95] Jacques Heyman. *The Stone Skeleton: Structural Engineering of Masonry Architecture*. Cambridge University Press, 1995.
- [Hor87] B.K.P. Horn. Closed-form solution of absolute orientation using unit quaternions. *J. of the Opt. Society of America A*, 4:629–642, 1987.
- [Joh74] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer a. System Sciences*, 9:256, 1974.
- [JTT<sup>+</sup>15] Caigui Jiang, Chengcheng Tang, Marko Tomičić, Johannes Wallner, and Helmut Pottmann. Interactive modeling of architectural freeform structures: Combining geometry with fabrication and statics. In Philippe Block, Jan Knippers, Niloy J. Mitra, and Wenping Wang, editors, *Advances*



- in Architectural Geometry 2014*, pages 95–108. Springer International Publishing, 2015.
- [JWWP14] Caigui Jiang, Jun Wang, Johannes Wallner, and Helmut Pottmann. Freeform honeycomb structures. *Computer Graphics Forum*, 33(5):185–194, 2014.
- [KBLK14] Michael Kremer, David Bommes, Isaak Lim, and Leif Kobbelt. Advanced automatic hexahedral mesh generation from surface quad meshes. In Josep Sarrate and Matthew Staten, editors, *Proceedings of the 22nd International Meshing Roundtable*, pages 147–164. Springer International Publishing, 2014.
- [KCPS13] Felix Knöppel, Keenan Crane, Ulrich Pinkall, and Peter Schröder. Globally optimal direction fields. *ACM Trans. Graph.*, 32(4), 2013.
- [Kil06] Axel Kilian. *Design exploration through bidirectional modeling of constraints*. PhD thesis, Massachusetts Institute of Technology, 2006.
- [KMP07] Martin Kilian, Niloy J. Mitra, and Helmut Pottmann. Geometric modeling in shape space. *ACM Trans. Graph.*, 26(3), July 2007.
- [KO05] Axel Kilian and John Ochsendorf. Particle-spring systems for structural form finding. *Journal of the International Association for Shell and Spatial Structures*, 46(2):77–84, 2005.
- [LFL09] Kui-Yip Lo, Chi-Wing Fu, and Hongwei Li. 3D Polyomino puzzle. *ACM Trans. Graph.*, 28(5):157:1–157:8, 2009.
- [Liv92] R. K. Livesley. A computational model for the limit analysis of three-dimensional masonry structures. *Meccanica*, 27(3):161–172, 1992.
- [LLSZ13] Qin Lyu, Zhouchen Lin, Yiyuan She, and Chao Zhang. A comparison of typical  $l_p$  minimization algorithms. *Neurocomput.*, 119:413–424, 2013.
- [LLW15] Yufei Li, Yang Liu, and Wenping Wang. Planar hexagonal meshing for architecture. *Visualization and Computer Graphics, IEEE Transactions on*, 21(1):95–106, Jan 2015.
- [LLX<sup>+</sup>12] Yufei Li, Yang Liu, Weiwei Xu, Wenping Wang, and Baining Guo. All-hex meshing using singularity-restricted field. *ACM Trans. Graph.*, 31(6):177:1–177:11, November 2012.
- [LPS<sup>+</sup>13] Yang Liu, Hao Pan, John Snyder, Wenping Wang, and Baining Guo. Computing self-supporting surfaces by regular triangulation. *ACM Trans.*

- Graph.*, 32(4):92:1–92:10, 2013.
- [LPW<sup>+</sup>06] Yang Liu, Helmut Pottmann, Johannes Wallner, Y.-L Yang, and Wenping Wang. Geometric modeling with conical meshes and developable surfaces. *ACM Trans. Graphics*, 25(3):681–689, 2006.
- [LXW<sup>+</sup>11] Yang Liu, Weiwei Xu, Jun Wang, Lifeng Zhu, Baining Guo, Falai Chen, and Guoping Wang. General planar quadrilateral mesh design using conjugate direction field. *ACM Trans. Graph.*, 30(6):140:1–140:10, December 2011.
- [MGP07] Niloy J. Mitra, Leonidas J. Guibas, and Mark Pauly. Symmetrization. *ACM Trans. Graphics*, 26(3):# 63, 2007.
- [Mos14] Mosek. The MOSEK optimization software. <http://www.mosek.com>, 2014.
- [NPPZ12] Matthias Nieser, Jonathan Palacios, Konrad Polthier, and Eugene Zhang. Hexagonal global parameterization of arbitrary surfaces. *IEEE Transactions on Visualization and Computer Graphics*, 18(6):865–878, June 2012.
- [NW06] Jorge Nocedal and Stephen Wright. *Numerical Optimization*. Springer Series in Operations Research and Financial Engineering. Springer-Verlag New York, 2nd edition, 2006.
- [Pan15] Daniele Panozzo. Demystifying quadrilateral remeshing. *Computer Graphics and Applications, IEEE*, 35(2):88–95, Mar 2015.
- [PBSH13] Daniele Panozzo, Philippe Block, and Olga Sorkine-Hornung. Designing unreinforced masonry models. *ACM Trans. Graph.*, 32(4):91:1–91:12, 2013.
- [PEVW15] Helmut Pottmann, Michael Eigensatz, Amir Vaxman, and Johannes Wallner. Architectural geometry. *Computers & Graphics*, 47(0):145 – 164, 2015.
- [PHD<sup>+</sup>10] Helmut Pottmann, Qixing Huang, Bailin Deng, Alexander Schiftner, Martin Kilian, Leonidas Guibas, and Johannes Wallner. Geodesic patterns. *ACM Trans. Graph.*, 29(4):43:1–43:10, 2010.
- [Pik13] Daniel Piker. Kangaroo: Form finding with computational physics. *Architectural Design*, 83(2):136–137, 2013.
- [PLW<sup>+</sup>07] Helmut Pottmann, Yang Liu, Johannes Wallner, Alexander Bobenko, and Wenping Wang. Geometry of multi-layer freeform structures for architecture. *ACM Trans. Graph.*, 26(3), 2007.
- [PSB<sup>+</sup>08] Helmut Pottmann, Alexander Schiftner, Pengbo Bo, Heinz Schmiedhofer,

- Wenping Wang, Niccolo Baldassini, and Johannes Wallner. Freeform surfaces from single curved panels. *ACM Trans. Graph.*, 27(3):76:1–76:10, 2008.
- [PW01] Helmut Pottmann and Johannes Wallner. *Computational Line Geometry*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2001.
- [RLB12] Matthias Rippmann, Lorenz Lachauer, and Philippe Block. Interactive vault design. *International Journal of Space Structures*, 27(4):219–230, 2012.
- [Rob15] Christopher Robeller. *Integral Mechanical Attachment for Timber Folded Plate Structures*. PhD thesis, ENAC, Lausanne, 2015.
- [ROR<sup>+</sup>10] Michael H. Ramage, John Ochsendorf, Peter Rich, James K. Bellamy, and Philippe Block. Design and construction of the Mapungubwe national park interpretive centre, South Africa. *African Technology Development Forum*, 7(1):14–23, 2010.
- [SFCO12] Peng Song, Chi-Wing Fu, and Daniel Cohen-Or. Recursive interlocking puzzles. *ACM Trans. Graph.*, 31(6):128:1–128:10, 2012.
- [She02] Dennis Shelden. *Digital surface representation and the constructibility of Gehry’s architecture*. PhD thesis, M.I.T., 2002.
- [SHWP09] Alexander Schiftner, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. Packing circles and spheres on surfaces. *ACM Trans. Graph.*, 28(5):139:1–139:8, 2009.
- [SMB14] Daniel Sieger, Stefan Menzel, and Mario Botsch. Constrained space deformation for design optimization. *Procedia Engineering*, 82(0):114 – 126, 2014. 23rd International Meshing Roundtable (IMR23).
- [SP13] Yuliy Schwartzburg and Mark Pauly. Fabrication-aware design with intersecting planar pieces. *Comput. Graph. Forum*, 32(2):317–326, 2013.
- [SP15] Gennaro Senatore and Daniel Piker. Interactive real-time physics: An intuitive approach to form-finding and structural analysis for design and education. *Computer-Aided Design*, 61(0):32–41, 2015.
- [SS10] Mayank Singh and Scott Schaefer. Triangle surfaces with discrete equivalence classes. *ACM Trans. Graph.*, 29(4):46:1–46:7, 2010.
- [STC<sup>+</sup>13] Mélina Skouras, Bernhard Thomaszewski, Stelian Coros, Bernd Bickel, and Markus Gross. Computational design of actuated deformable characters.

- ACM Trans. Graph.*, 32(4):82:1–82:10, 2013.
- [SVB<sup>+</sup>12] Ondrej Stava, Juraj Vanek, Bedrich Benes, Nathan Carr, and Radomír Měch. Stress relief: Improving structural strength of 3D printable objects. *ACM Trans. Graph.*, 31(4):48:1–48:11, 2012.
- [TC89] S.M. Thomas and YT Chan. A simple approach for the estimation of circular arc center and its radius. *Computer Vision, Graphics, and Image Processing*, 45:362–370, 1989.
- [TSG<sup>+</sup>14] Chengcheng Tang, Xiang Sun, Alexandra Gomes, Johannes Wallner, and Helmut Pottmann. Form-finding with polyhedral meshes made simple. *ACM Trans. Graph.*, 33(4):70:1–70:9, 2014.
- [Ume91] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *Pattern Analysis and Machine Intelligence*, 13:376–380, 1991.
- [US13] Nobuyuki Umetani and Ryan Schmidt. Cross-sectional structural analysis for 3D printing optimization. In *SIGGRAPH Asia 2013 Technical Briefs*, 2013.
- [VHWP12] Etienne Vouga, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. Design of self-supporting surfaces. *ACM Trans. Graph.*, 31(4):87:1–87:11, 2012.
- [VMMDB12] Tom Van Mele, James McInerney, Matthew DeJong, and Philippe Block. Physical and computational discrete modeling of masonry vault collapse. In *Proc. Int. Conf. Structural Analysis of Historical Constructions*, 2012.
- [WB97] Andrew Witkin and David Baraff. Physically based modeling: Principles and practice. Siggraph '97 Course notes, 1997.
- [Wen09] David Wendland. Experimental construction of a free-form shell structure in masonry. *International Journal of Space Structures*, 24(1):1–11, 2009.
- [WOD09] Emily Whiting, John Ochsendorf, and Frédo Durand. Procedural modeling of structurally-sound masonry buildings. *ACM Trans. Graph.*, 28(5), 2009.
- [WP06] Yves Weinand and Claudio Pirazzi. Geodesic Lines on Free-Form Surfaces - Optimized Grids for Timber Rib Shells. In *World Conference in Timber Engineering WCTE*, 2006.
- [WSW<sup>+</sup>12] Emily Whiting, Hijung Shin, Robert Wang, John Ochsendorf, and Frédo Durand. Structural optimization of 3D masonry buildings. *ACM Trans.*

- Graph.*, 31(6):159:1–159:11, 2012.
- [WWY<sup>+</sup>13] Weiming Wang, Tuanfeng Y. Wang, Zhouwang Yang, Ligang Liu, Xin Tong, Weihua Tong, Jiansong Deng, Falai Chen, and Xiuping Liu. Cost-effective printing of 3D objects with skin-frame structures. *ACM Trans. Graph.*, 32(5):177:1–177:10, 2013.
- [XLF<sup>+</sup>11] Shiqing Xin, Chi-Fu Lai, Chi-Wing Fu, Tien-Tsin Wong, Ying He, and Daniel Cohen-Or. Making burr puzzles from 3D models. *ACM Trans. Graph.*, 30(4):97:1–97:8, 2011.
- [YLW06] Dong-Ming Yan, Yang Liu, and Wenping Wang. Quadric surface extraction by variational shape approximation. In *Proceedings of the 4th International Conference on Geometric Modeling and Processing*, GMP’06, pages 73–86, Berlin, Heidelberg, 2006. Springer-Verlag.
- [YYPM11] Yong-Liang Yang, Yi-Jun Yang, Helmut Pottmann, and Niloy J. Mitra. Shape space exploration of constrained meshes. *ACM Trans. Graph.*, 30(6):124:1–124:12, December 2011.
- [ZCBK12] Henrik Zimmer, Marcel Campen, David Bommes, and Leif Kobbelt. Rationalization of triangle-based point-folding structures. *Comp. Graph. Forum*, 31(2pt3):611–620, May 2012.
- [ZLAK14] Henrik Zimmer, Florent Lafarge, Pierre Alliez, and Leif Kobbelt. Zometool shape approximation. *Graphical Models*, 76(5):390–401, 2014.
- [ZLO10] Jennifer Zessin, Wanda Lau, and John Ochsendorf. Equilibrium of cracked masonry domes. *Proc. ICE-Engineering and Computational Mechanics*, 163(3):135–145, 2010.
- [ZSW10] Mirko Zadravec, Alexander Schiftner, and Johannes Wallner. Designing quad-dominant meshes with planar faces. *Computer Graphics Forum*, 29(5):1671–1679, 2010.
- [ZTY<sup>+</sup>13] Xin Zhao, Cheng-Cheng Tang, Yong-Liang Yang, Helmut Pottmann, and Niloy J. Mitra. Intuitive design exploration of constrained meshes. In Lars Hesselgren, Shrikant Sharma, Johannes Wallner, Niccolo Baldassini, Philippe Bompas, and Jacques Raynaud, editors, *Advances in Architectural Geometry 2012*, pages 305–318. Springer Vienna, 2013.



# Curriculum Vitae

## Mario Deuss

**Contact:** [mario.deuss@gmail.com](mailto:mario.deuss@gmail.com), +41 77 436 62 00,  
Martastrasse 136, 8003 Zurich, Switzerland



### Technical skills:

Languages	C/C++ (Qt, OpenGL, Eigen, OpenMesh), GLSL, Matlab
Topics	<ul style="list-style-type: none"><li>- Geometry processing (remeshing, parametrization, deformation, modeling)</li><li>- Computer graphics (real-time rendering and interaction, shaders, path tracing)</li><li>- Numerical optimization (real-time constrained optimization, sparsity objectives)</li></ul>
Software	QtCreator, Visual Studio, 3dsmax, Blender, Illustrator, Svn, Git, Homebrew, OS X

### Employment:

Feb '10 – Aug '15	PhD at LGG, EPF Lausanne: <ul style="list-style-type: none"><li>- Developed and maintained a geometry processing software of 800k lines of C++ code using OpenGL, Qt, OpenMesh and Eigen</li><li>- Implemented numerical optimizations, in particular Non-linear Least-squares and Augmented Lagrangian with various objective functions and derivatives using sparse matrices in C++ and Eigen</li><li>- Developed part of ShapeOp (<a href="http://www.shapeop.org">www.shapeop.org</a>), an open-source C++ library for geometry processing, designed the C API and used SWIG to generate wrappers</li></ul>
Feb '10 – now	Teaching assistant at EPF Lausanne for the lectures: "Introduction to Computer Graphics", "Advanced Computer Graphics", "Digital 3D Geometry Processing" and "Linear Algebra". Designed various programming exercises, including a Monte Carlo path tracer, in C++, GLSL and Python. Supervising 5 Semester and Master Theses.
Jun '13 – Aug '13	Research Internship at Microsoft Research Asia, Beijing, China. Studied non-vertical loads and assembly of self-supporting surfaces. This internship lead to my publication "Assembling Self-Supporting Structures" at SIGGRAPH ASIA 2014.
Jan '08 – Jul '08	Research Assistant of Prof. Mark Pauly with ETH & Undisclosed Industrial Partner, Zurich. Implemented software for real-time surface texturing using particles and local parametrization in C++, OpenMesh, OpenGL and Qt4.
Jul '07 – Dez '07	Internship at Autoform GmbH, Zurich, Switzerland. Analyzed and redesigned an automatic detection of production-relevant geometric features and their cost factors from C to C++.

**Education:**

Feb '10 – Aug'15	PhD at Computer Graphics and Geometry Laboratory at EPF Lausanne
Apr '09 – Sep '09	Master's Thesis in Computer Science at the Stanford University, Palo Alto, USA
2004 – 2009	BSc and MSc (GPA: 5.60 out of 6.00) at ETH Zurich, Department Computer Science

**Publications:**

2015	"ShapeOp - A Robust and Extensible Geometric Modelling Paradigm", M. Deuss, A. H. Deleuran, S. Bouaziz, B. Deng, D. Piker, M. Pauly, Design Modelling Symposium 2015
2014	"Assembling Self-Supporting Structures", M. Deuss, D. Panozzo, E. Whiting, Y. Liu, P. Block, O. Sorkine-Hornung, M. Pauly, SIGGRAPH ASIA 2014
2014	"Interactive Design Exploration for Constrained Meshes", B. Deng, S. Bouaziz, M. Deuss, A. Kaspar, Y. Schwartzburg, M. Pauly, CAD 2014
2013	"Exploring Local Modifications for Constrained Meshes", B. Deng, S. Bouaziz, M. Deuss, J. Zhang, Y. Schwartzburg, M. Pauly, Eurographics 2013
2012	"Shape-Up: Shaping Discrete Geometry with Projections", S. Bouaziz, M. Deuss, Y. Schwartzburg, T. Weise, M. Pauly, SGP 2012
2010	"Case Studies in Cost-Optimized Paneling of Architectural Freeform Surfaces", M. Eigensatz, M. Deuss, M. Kilian, N. Mitra, H. Pottmann, M. Pauly, AAG 2010

**Personal:**

Date of birth:	January 19 <sup>th</sup> , 1984
Citizenship:	Swiss, Brazilian
Languages:	German (mother tongue), English (fluent), French (fluent), Portuguese (basics)
Hobbies:	Guitar, Singing, Dancing, Skate- and Snowboarding
Extracurricular:	Leader of a scout group of 60 people from 1999 until 2002, in particular a two weeks summer camp in 2002.