Pushing the Limits of Additive Fabrication Technologies

by

Qingnan Zhou

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy Department of Computer Science New York University May, 2016

Denis Zorin

Dedication

This is thesis is dedicated to my parents and my wife.

Mom and Dad, thanks for supporting my decision of giving up offers from Google and Amazon in order to pursue my passion even if that caused delayed retirement and financial hardship. Thanks for believing in me and my potentials in both good days and bad days.

Fangchun, thanks for staying by my side and encouraging me during every single Siggraph push. Thanks for bringing me food, water and chocolate when I have to work overnight. I cannot make it without you. You are the best wife I can ask for.

Acknowledgements

I would love to thank my supervisor Denis Zorin for introducing me to the world of 3D printing and for his continuous support and patience. I also thank all my collaborators: Julian Panetta, Alec Jacobson, Eitan Grinspun, Nico Pietroni, Luigi Malomo, Paolo Cignoni, David Harmon, Tino Weinkauf and Olga Sorkin-Hornung. I would have not made it without them.

I also want to thank my industry collaborators for providing me with test samples and 3D printers free of charge: Shapeways, Autodesk, FormLabs, Carbon, ExOne, nTopology, Amazon and Laguardia Studio. I also thank Prof. Yu Zhang from NYU Dental School, Prof. Oran Kennedy from NYU School of Medicine and Prof. Nikhil Gupta from NYU Tendon School of Engineering for allowing me to use their universal testing machines to conduct material testing even on weekend and holidays.

Lastly, I would like to thank my friends and coworkers from whom I learned so much: Ashish Myles, Denis Kovacs, Daniele Panozzo, Francisca Gil Ureta, Chelsea Tymms, Eduardo Corona, Johnathan Thompson, Murphy Stein, Ross Goroshin, Abtin Rahimian, Ofir Weber, Li Wan, Sixin Zhang, Junjie Chen, Wei Wang, Nancy Yi Liang and Mary Huang.

Abstract

In this thesis, I will present the results of three projects addressing some of the major chanllenges of applying computational design for 3D printing.

Printability test has long been the bottleneck that prevents 3D printing from scaling up. Oftentimes, designers of 3D models lack the expertise or tools to ensure 3D printability. 3D printing service providers typically rely human inspections to filter out unprintable designs. This process is manual and error-prone. As designs become ever more complex, manual printability check becomes increasingly difficult. To tackle this problem, my colleagues and I proposed an algorithm to automatically determine structurally weak regions and the worst-case usage scenario to break a given model. We validate the algorithm by physically break a number of real 3D printed designs.

The second project focuses on designing, analyzing and fabricating complex shapes. A key distinctive feature of 3D printing technologies is that the cost and time of fabrication is uncorrelated with geometric complexity. This opens up many exciting new possibilities. In particular, by pushing geometric complexity to the extreme, 3D printing has the potential of fabricating soft, deformable shapes with microscopic structures using a single raw material. In our recent SIGGRAPH publication, my colleagues and I have not only demonstrated fabricating microscopic frame structures is possible but also proposed an entire pipeline for designing spatially varying microstructures to satisfy target material properties or deformation goals.

An essential component of computational design for 3D printing is elasticity simulation, which requires meshes satisfying a number of criteria. With the boost of 3D printing technologies, 3D models have become more abundant and easily accessible than ever before. However, a significant portion of these models violate such criteria. This poses a serious challenge in robustly analyzing 3D designs. Many state-of-the-art geometry processing algorithms/libraries are ill-prepared for dealing with models that are nonmanifold, self-intersecting, locally degenerate and/or containing multiple and possibly nested components. In our most recent SIGGRAPH publication, we proposed an algorithm based on mesh arrangements for conducting a family of exact constructive solid geometry operations. We exhaustively tested our algorithm on 10,000 models crawled from Thingiverse, a popular online shape repository. Both the code and the dataset are freely available to the public.

Contents

D	edica	tion	ii	
A	cknov	wledgements	iii	
A	bstra	\mathbf{ct}	iv	
\mathbf{Li}	st of	Figures	ix	
Li	st of	Tables	xviii	
1	Intr	oduction	1	
	1.1	Background	. 2	
		1.1.1 Technologies	. 2	
	1.2	Challenges and Research Opportunities	. 5	
		1.2.1 Repeatability	. 5	
		1.2.2 Failure rate	. 6	
		1.2.3 Scaling up	. 6	
	1.3	Contributions	. 6	
2	Wor	rst-case Structrual Analysis	8	
	2.1	Introduction	. 8	
	2.2	Related work	. 10	
		2.2.1 3D printing processes	. 12	
	2.3	Worst-case structural analysis	. 14	
	2.4	Efficient approximate algorithm	. 18	

	2.5	Validation	26
	2.6	Material properties	35
	2.7	Conclusions	38
3	Elas	stic Textures for Additive Fabrication 4	10
	3.1	Introduction	1 1
	3.2	Related Work	43
	3.3	Overview and Main Results	4 6
	3.4	Search for Efficient Patterns	53
	3.5	From Patterns to Material Properties	59
	3.6	Optimizing Pattern Parameters	33
	3.7	Applications	37
	3.8	Conclusions	70
	ъл		
4	Mes	Sh Arrangement for Solid Geometry	2
	4.1	Introduction	72
	4.2	Related work	74
	4.3	Concepts	30
		4.3.1 Piecewise-constant winding number meshes	30
		4.3.2 Variadic extraction function	33
		4.3.3 Solid meshes	34
	4.4	Overview	35
	4.5	Algorithms	36
		4.5.1 Intersection resolution	37
		4.5.2 Partitioning space into cells) 0
		4.5.3 Winding number labeling)4
		4.5.4 Operation result extraction)5
		4.5.5 Core low-level subroutines	96
	4.6	Implementation)1
		4.6.1 Converting to floating-point)2

	4.7	Experiments and results
		4.7.1 Thingi10K dataset
		4.7.2 Testing self-union
		4.7.3 General discussion
	4.8	Limitations & Future work
5	Thi	ngi10K: A Dataset of 10,000 3D-Printing Models 119
	5.1	Introduction and background
	5.2	Methodology
	5.3	Analysis
		5.3.1 Geometry information
		5.3.2 Contextual information
	5.4	Online query interface
	5.5	Conclusion
6	Cor	nclusion 136
	6.1	Unsolved problems and future works
		6.1.1 Material properties
		6.1.2 Accuracy
		6.1.3 Volumetric meshing
		6.1.4 Large scale validation
Bi	ibliog	graphy 139

List of Figures

2.1	Algorithm overview	8
2.2	The top 10% volume of largest principal stress (left) and largest trace	
	(right) are visually similar	21
2.3	Histogram of the mode number (horizontal axis) in which the weakest	
	region appears for the first time	25
2.4	Histogram of the rank of the weakest region in the weak region list sorted	
	by decreasing energy	25
2.5	Weak regions extracted from three modes with weakness level cutoff, $\epsilon =$	
	.10, .05, .03, .01	27
2.6	Optimal force vectors and weakest regions on the left, resulting deforma-	
	tions and stresses on the right. The gray images in the background show	
	the undeformed state	27
2.7	Comparison of the similar optimum stresses found by brute force, Laplacian-	
	based weak region analysis, and stiffness-based weak region analysis. The	
	table reports 99.75 percentile (by volume) element stresses. An isotropic	
	metal material was used for this comparison. \ldots . \ldots . \ldots . \ldots	28
2.8	For 5 different mesh resolutions (from left to right, the vertex counts are	
	5K, 13K, 24K, 36K, 50K), the algorithm generates consistent weakness	
	maps	29
2.9	We used models printed in green state "sandstone" for the drop tests. The	
	testing models often are covered with a loose layer of powder that shakes	
	off upon impact (see the dust in the right image). \ldots \ldots \ldots \ldots	30

2.10	Results for a drop test. Model volume is shaded with its weakness map	
	percentile: 90% 99%	31
2.11	Simulation results (left) are compared to the deformed 3D printed model	
	(green) overlaid on an undeformed one (blue). Our algorithm predicts	
	likely regions (red) of large deformation under normal handling. For the	
	blade earring, we confirm that the largest blade deforms little relative to	
	the hook and shaft: after aligning the shafts to be parallel, the largest	
	blades are also roughly parallel (see the yellow parallelograms). The sec-	
	ond largest blade is displaced more. Note that the upper right pin of the	
	deformed spinnoloid (middle row, green) was broken during printing. $\ .$.	32
2.12	2 Models where pinch grips cannot generate worst-case loads. Our method	
	finds highly intuitive force vectors, regardless. The additional arrows on	
	the top of the Skyrim dragon arise to bring the total force and torque to	
	zero	33
2.13	Comparison against $[120]$. Our algorithm's force distribution (top) better	
	identifies structural weakness, especially for the ufo (middle) and bracelet	
	(right)	33
2.14	The top plots shows how modal analysis and weak region extraction scale	
	with the number of tetrahedra. The dominant cost is the eigensolve. The	
	bottom plots shows the average cost of setting up and running the linear	
	program for each weak region. It excludes the UMFPACK factorization	
	of C^* that only must be run once.	34
2.15	Model vertex counts tabulated from 2781 models ordered from Shapeways.	35
2.16	5 Different ratios of directional Young's moduli can lead to different weakest	
	regions. We show the weakest region found for a truss with a Young's	
	modulus that is five times higher in the X (left), Y (middle), and Z (right)	
	directions	36
2.17	Three-point bending test on green state stainless steel	36

2.18	Left: Stress vs strain curve measured on samples in green state stainless	
	steel. The colors indicate different sample thickness (1.5mm red, $2mm$	
	green, 3mm blue). Right: Stress vs strain plots for nylon testing samples	
	of thickness 1.5mm and 2mm. The samples printed in different orientation	
	are marked with different colors (red: X, green: Y, blue: Z). \ldots .	37
2.19	The critical stress distribution of green state metal for samples with thick-	
	ness 1.5mm up to 5mm	37
2.20	Stress vs. strain measurements on rectangular bars printed with green	
	state "sandstone" along the printer X (red), Y (green) and Z (blue) di-	
	rection. Different printing directions influence the material properties	
	significantly.	38
२ 1	Six basic elastic textures are used to obtain a large range of homogenized	
0.1	Six subjections are used to obtain a large range of noningenized	
	isotropic material properties. A $3 \times 3 \times 1$ tiling of each pattern is shown,	
	along with rendered (left) and fabricated (right) cell geometry below. The	
	naming convention is explained in Section 3.4	41
3.2	Overview of elastic texture generation and use	46
3.3	Region of the (E, ν) space covered by the selected set of patterns. Each	
	topology's coverage is shown in a different color. \ldots \ldots \ldots \ldots \ldots	50
3.4	Samples of the (E, ν) space reached by patterns with topology "(E1,E2)	
	(E1,E4) $(E2,E4)$."	50
3.5	Compression test results for eight patterns with varying homogenized	
	Young's moduli (6 \times 6 \times 2 tiling of 5mm cells). (a) Slopes extracted	
	from the measured force vs. displacement curves along with a best-fit	
	line through the origin. (b) Moduli extracted from simulated compression	
	tests, with and without modeling compression plate friction. Without fric-	
	tion, the simulated test agrees with homogenization perfectly, but friction	
	introduces error.	51

3.6	Poisson's ratios measured from $3 \times 3 \times 1$ printed tilings of 10mm cells	
	vs. homogenized properties. The $\nu=-0.67$ sample, outside our family's	
	range, violates isotropy and printability constraints (we added support	
	structure manually for this experiment)	52
3.7	We extracted a 45° rotated rectangular block from a regularly tiled 10mm	
	cell microstructure to test Young's modulus in non-axis aligned directions.	52
3.8	(a) The tetrahedral cube decomposition used to generate 3D patterns;	
	(b) The 15 nodes defined on a tetrahedron together with their degrees of	
	freedom	54
3.9	Symmetry orbits are colored with yellow, red and green. Left: vertex	
	symmetry orbits. Right: edge symmetry orbits.	55
3.10	The results of varying the thickness (top) and offset (bottom) parameters	
	of a particular pattern topology.	55
3.11	Two pattern topologies from each of three different families, shown with	
	the families' <i>interfaces</i> (nodes on the cube cell faces)	56
3.12	2D examples of the printability detection algorithm. Vertices with sup-	
	porting nodes are marked (green), then a breadth-first search extends the	
	supported vertex front to horizontal neighbors. The remaining unmarked	
	nodes are unsupported (red). Two cases are shown: unprintable (top) and	
	printable (bottom)	57
3.13	(Schematic) Periodic tiling of a domain Ω with base cell Y having geom-	
	etry ω and length scale ϵ .	59
3.14	Deformation of an object with varying material properties per voxel, and	
	the same object with the material in each voxel replaced with the corre-	
	sponding pattern. The deformed objects are colored by max stress	62
3.15	Left: a shape derivative, visualized as a steepest ascent normal velocity	
	field for objective 3.8 . Right: the shape velocity induced by one of the	
	pattern's thickness parameters.	65

3.16	Convergence of a shape optimization on pattern "(E1,E2) (E1,E4) (E2,E4)."	
	Left: optimization starting point. Right: optimized shape	66
3.17	The path in (E,ν) space traversed by the optimization of pattern "(E1,E2)	
	(E1,E4) (E2,E4)" shown in Figure 3.16. The brown points are intermedi-	
	ate anisotropic microstructures	66
3.18	Examples of objects with painted material properties. All are fabricated	
	with 5mm cells	68
3.19	Convergence of material optimization.	69
3.20	Examples of objects with optimized material properties. All are fabricated	
	with 5mm cells	71
3.21	Compression of an anisotropic sample along the X, Y, and Z directions. .	71
4.1	Our method takes as input any number of meshes (three shown in this	
	2D illustration). We resolve intersections and assign a winding number	
	vector to every delineated cell. Different boolean operations are achieved	
	as extractions according to these winding number vectors	72
4.2	Self-intersections are not just "artifacts." They also occur intentionally	
	during modeling. A coffee mug handle is extruded then curled inward on	
	itself (blue). The self-intersections (orange) do not prohibit constructing	
	the shape's underlying torus-topology self-union with our algorithm (green).	75
4.3	Combinatorially, the multi-component shapes on the left and right are the	
	same, though their outer hulls (thick) differ	77
4.4	The outer-hull (e.g., $[9, 31]$ is simpler to extract, but not always appro-	
	priate. Hollow cavities should be maintained for 3D printing not only to	
	reduce material costs, but to maintain functional properties like balancing.	78
4.5	By outputting an exact solid mesh, our method ensures there are no in-	
	versions (illustrated as correct signed distance on left). Previous methods	
	may create inversions, and even small inversions catastrophically effect	
	signed distance to the output (right)	79

4.6	Our assumptions allow a wide class of inputs with self-intersections, multi-	
	ple components and non-manifold connections, but does not include open	
	boundaries or non-manifold flaps	31
4.7	Alternative extractions $(\mathcal{AB}, \mathcal{AB}, \mathcal{B} \setminus \mathcal{A})$ share the same input mesh ar-	
	rangement	36
4.8	Blue triangles intersect the green at a point, segments and a polygon	
	(<i>left, 3D</i>). Subdivided constraints reduce to coplanar points and segments	
	(middle, $2D$). The original green triangle is replaced with replications of	
	the green triangles of a constrained Delaunay triangulation (CDT) of the	
	coplanar cluster ($right$, $2D$)	38
4.9	$Two \ overlapping, \ co-planar \ right \ triangles \ (left) \ admit \ multiple \ constrained$	
	Delaunay triangulations (CDTs). Independent triangulation could lead to	
	inconsistent CDTs (middle and right)	39
4.10	We retain the relationship between the output triangles and the inputs.	
	Because of our exactness, attributes like texture coordinates are losslessly	
	maintained	39
4.11	Consider a tetrahedron (orange) inside a wedge (blue), connected at a non-	
	manifold vertex. One face normal of the interior tetrahedron (orange) has	
	a larger x -component than the normals of exterior triangles (blue) incident	
	on this non-manifold <i>outer vertex</i>)8
4.12	Consider five co-planar facets (lines) incident on two oriented edges (dots).	
	Using the edge orientation to $sign$ these indices during sorting ensures that	
	orderings from either edge are consistent (left). Otherwise the sort is the	
	same regardless of the edge orientation leading to inconsistent ordering. $% \left[{{\left[{{{\left[{{\left[{\left[{{\left[{{\left[{{\left[$)0
4.13	We conduct extensive evaluation of the robustness of our method on a	
	dataset of 10,000 popular real-world models. $\ldots \ldots \ldots$)1
4.14	The Thingi10K dataset contains real-world meshes with real-world problems.10)3

4.15	Previous methods frequently failed to produce an output without self-
	intersections, without open boundaries, and with piecewise-constant wind-
	ing number
4.16	The performance of our method is competitive with existing floating-point
	methods and faster than the state-of-the-art exact method [34] Geometric
	means given for each method. Unequal histogram areas correspond to
	success rate 107
4 17	The main bettlened, of our algorithm is triangle triangle intersection res
4.17	The main bottleneck of our algorithm is triangle-triangle intersection res-
	olution. Within this major stage, the intersection detection and exact
4.10	construction dominate
4.18	Self-intersections confuse per-vertex ambient occlusion and sharp-line de-
	tection. Rendering the outerhull ameliorates this
4.19	David emerges from a block by repeatedly subtracting the Minkowski sum
	of a drill bit along piecewise-linear paths
4.20	Numerical perturbation can produce spurious artifacts
4.21	We reproduce and exhaustively expand the pairwise testing in [16] (see
	supplemental material)
4.22	We reproduce and expand upon the \mathcal{A} -union-rotated- \mathcal{A} style test in [16] 112
4.23	A popular commercial app produces three different results (presumably
	due to randomization), yet all are incorrect
4.24	Our method supports n -ary operations. For example, extracting all re-
	gions inside at least k of the input spheres centered at each corner of the
	unit cube
4.25	Simple and complex variadic operations cost the same using our mesh
	arrangements. Converting variadic operations to a cascade of binary op-
	erations is worst-case exponential in time
4.26	Disconnected stars and overlapping spike components correctly unites
	with a nested turtle (slice view above)

4.27	We reproduce the carving example of [20] by subtracting 10,000 dodeca-
	hedra from a box
4.28	³ The groved, yellow frog is the result of subtracting the stripy, blue flog
	from an unknown (presumably solid) frog using some boolean implemen-
	tation (not ours). Our robust union recovers the original frog (blue and
	yellow)
4.29	Tetrahedralization of the input foot mesh fails due to overlapping input
	components, but succeeds after self-union. The volume mesh helps analyze
	the shapes structure
4.30	The coin mesh has been modeled with many overlapping components
	(gold). Constructing the outer hull adds many new vertices, but removes
	hidden interior geometry. After decimation, the few triangles are well
	spent on the visible surface (silver). Naive decimation wastes precious
	triangles on the hidden interior (bronze)
4.31	The letters are separated overlapping components on the cryptex. Exact
	union reveals disjoint rings, shown in different colors, but small tolerances
	between parts cause inexact methods to merge over zealously 118
4.32	2 The self-union of a wheel of cheese retains its internal bubbles after slicing
	(intersecting) with a knife (blue wedge). Slicing the outer hull reveals no
	bubbles. Eliminating small volumes cells in the self-union before extrac-
	tion, produces a few bubbles
5.1	The Thingi10K dataset contains 10,000 models from from featured "things"
	on thingiverse.com, a popular online repository
5.2	Our online query interface selects subsets of Thingi10K
5.3	Percentile plots of vertex and component count
5.4	Highest resolution models from each dataset

5.5	Connected components of Thingi10K models tend to represent salient
	parts; those in ShapeNetCore are often just disconnected patches (models
	with most components shown)
5.6	Models with the highest genus from each dataset
5.7	MPZ14 models are "too clean," whereas ShapeNetCore are unrealistically
	corrupted in the context of 3D printing. $\ldots \ldots \ldots$
5.8	Percentile plots mesh quality measures
5.9	Models with tag math (left), sculpture (middle) and scan (right). \dots 130
5.10	Thingi10K user tags highlight the dataset's variety
5.11	A soap bubble chair is decomposed and re-oriented by its designer for
	support-free 3D printing
5.12	All 10,000 models come under open source licenses
5.13	Both contextual and geometric information of each model are available on
	its model detail page
5.14	Our web interface returns subsets of the Thingi10K dataset via text queries.
	135

List of Tables

2.1	Stress analysis timings for brute force optimization vs. weak region op-	
	timization. While speedups are already dramatic for extremely small el-	
	ement counts, the higher asymptotic complexity of brute force causes a	
	rapidly increasing speedup for larger models	28
4.1	Mesh boolean algorithm input preconditions feature chart: Previous tech-	
	niques fall short have severe input restrictions.	79

Chapter 1

Introduction

During the past 5 years, 3D printing technologies have gained tremendous momentum both in industry and academic research.

On the industry side, costs of 3D printers have dropped considerably. The expiration of Fused Deposition Modeling (FDM) patent has led to renewed interests in 3D printing from open source and maker communities. In addition, the emergence of 3D printing service providers has democratised 3D printing by making multi-million dollar 3D printers affordable and accessible by everyone. New generation of companies such as FormLabs and Carbon3D are constantly challenging the status quo on frontiers ranging from aesthetics, user experience to print quality and speed.

On the academic side, computational design and analysis for fabrication purposes have attracted the attention of researchers from different fields, ranging from mechanical engineering to mathematics. Not only do 3D printing technologies bring new challenges (Chapter 2) and possibilities (Chapter 3), they also serve as touch stones on state-of-theart geometry processing algorithms (Chapter 4) and highlight the gaps between academic research and industrial requirements (Chapter 5).

I am lucky enough to have conduct my research during this time, and I hope my work

presented in this thesis will make an impact to the development of 3D printing technologies.

1.1 Background

1.1.1 Technologies

3D printing (a.k.a. additive manufacturing) is a broad term referring to a number of distinct technologies with the common trait that they build up a given solid shape in a layer-wise manner. A survey of all 3D printing technologies is beyond the scope of this thesis. I refer interested readers to [69] and [138] for complete reviews of state-of-the-art methods. In this work, I focus on three printing techniques that I have worked with: filament extrusion, powder binding/sintering and photopolymer solidification.

Filament extrusion

There are many names for filament extrusion based 3D printing technologies: Fused Deposition Modeling (FDM), Fused Filament Fabrication (FFF), Plastic Jet Printing (PJP), etc. Among them, FDM is most commonly used.

FDM uses different types of thermoplastics filaments as raw material, a material that becomes soft and viscous at high enough temperature. To creates a layer of a target 3D shape, FDM printer extrudes melted material and uses it as paint to both trace the boundary and draw a 2D pattern (called infill) to fill its cross section. A sparse infill allows FDM printers to fabricate a given shape with very little raw material.

FDM printers are relatively easy to operate at very low cost. This makes them ideal for individual designers, hobbyists and educators. Some FDM printers support printing with multiple filaments in one print, giving designer more freedom in terms of mixing color and softness in their design. An important implication of multi-material FDM printers is that support structures, if needed, can be printed in a different material than the main pieces. If the support is printed with a soluble material, it is possible to fabricate complex design without worrying about ways of removing support structures.

FDM techniques also have a number of constraints. The printed objects often have very visible layer lines [99]. Even with solid infill, the printed materials are very anisotropic [2]. Typically, users choose a sparse infill pattern, which causes the material properties to be more complex to analyze. FDM is also geometrically less accurate than other techniques [80], making it unsuitable for fabricating parts with high accuracy requirement.

Powder binding/sintering

Powder-based 3D printing technologies are one of the most popular ways of fabricating customized designs at large scale. It is a broad term that encompasses a range of different techniques including Selective Laser Sintering (SLS), Direct Metal Laser Sintering (DMLS), Electron Beam Melting (EBM), Selective Laser Melting (SLM), and binder jetting (confusingly abbreviated as 3DP).

All these different technologies share the same work flow: (1) A thin layer of material powder is distributed on a flat surface; (2) A technique is used to merge the powders in certain regions; (3) The built platform is lowered and a new layer of powders is distributed. What distinguish these technologies is how powders are merged in step 2. DMSL and SLS use laser to sinter powders so each grain merge with neighboring grains. EBM and SLM use electron beam and laser respectively to completely melt the grains. Lastly, 3DP deposits special binder to glue the grains together. With the exception of EBM and SLM, the printed shapes are porous and have a rough surface finish. Post-processing is often necessary to fill in the pores and re-enforce the structural strength.

The major advantage of powder-based printing technologies is that no support structure is necessary because the loose powders from previous layers serve as supports for the current layer. This advantage allows powder-based printing to scale up and print multiple shapes simultaneously. Each run of the printer could fabricate tens or hundreds of shapes. In addition, due to the usage of binder, 3DP technology could be apply to a variety of materials, but post processing may be needed.

The powder form of the raw material also has its own drawbacks. The porous nature of the prints making material properties hard to predict. Two prints of the same shape may exhibit different structural strength. Similar to filament-based printing methods, the material properties of the output tend to be anisotropic. Another drawback is that the printed shapes need to be dug out from a block of powders. This process has to be conducted manually, thus error-prone. For binder jetting technologies, the raw prints (also known as "green parts"), are brittle and could break easily during digging and cleaning.

Photopolymer Solidification

Unlike filament extrusion and powder binding/sintering technologies, photopolymer solidification works with raw material in liquid form. The raw material, photopolymer resin, undergoes cross-linking when exposed to light, which causes it to transform from liquid state to solid state.

Two of the most popular photopolymer solidification technologies are Stereolithography (SLA) and Digital Light Processing (DLP). SLA uses high powder laser beams to solidify a layer by tracing the interior of a cross section of the target shape. In contrast, DLP rely on a conventional projector as light source to cure the entire cross section all at the same time.

Photopolymer solidification technologies are more accurate than most filament extrusion and powder binding/sintering technologies ([80]). [6, 143] demonstrate that hair-like features can be fabricated. SLA and DLP are widely used in jewelry and dental industries due to their high accuracy. Because the raw material is in liquid form, it is often messy to work with. Once a part is printed, the user needs to remove any uncured resin by rinsing with special chemicals. If a part failed to be printed, little fragments of cured layers would be scattered all over the place in the resin tray, which requires a thorough cleaning. Lastly, some printers rely a thin layer of Polydimethylsiloxane (PDMS) to allow both light and oxygen to come into contact with resin. This layer of PDMS usually gets cloudy over repeated usages and causes the print quality to deteriorate.

Another drawback of SLA and DLP is that the material properties changes over time. Sometimes, resin do not solidify fully during curing. Printed shapes may gradually become less flexible and brittle over time. Post curing the prints in UV box alleviates but does not eliminate this issue.

1.2 Challenges and Research Opportunities

During my PhD studies, I have the pleasure of meeting, talking to and working with many people working in different sectors of the 3D printing ecosystem. Their insights and experiences are invaluable for me in my current and future research. Here I summarize some of the most mentioned challenges that I have learned from them.

1.2.1 Repeatability

Repeatability is the number one issue users complain about 3D printing. It is especially frustrating if a previously printed design failed to be printed or rejected by 3D printing service providers. In addition to human errors, subtle changes such as an aging PDMS, over-recycled raw material, inaccurate calibration or even different room temperature or humidity could cause a previously printed design fail to print again. 3D printer manufacturers are aware of these issues and are actively making progress in improving printer reliability.

1.2.2 Failure rate

Like any early stage technologies, 3D printing fails a lot. Even experienced 3D designers report failure rates ranging from 30% to 90% before getting a design successfully printed. Some of these failures are catastrophic, with nothing coming out of the printer. Other failures include artifacts, missing details or unfunctional parts. Designers have to either adjust their design or adjust the printer setting and try again.

1.2.3 Scaling up

Unlike traditional manufacturing, 3D printing excels at creating customized, one-off models. While it is possible for individual users to print one model at a time, it is infeasible for 3D printing service providers to do so. In order to scale up and increase throughput, industrial 3D printing often group multiple models into a batch and print them together. One problem arise from this practice is that a single bad input may ruin the entire batch. Very few printers has a mature fault detecting and recovering system in place. Thus it is especially crucial for 3D printing service providers to analyze each model for printability before actually printing it.

1.3 Contributions

While the causes of these challenges may be diverse and complex, it is conceivable that all of them can be improved by incorporating appropriate computational design and analysis in the 3D printing work flow. This thesis summarizes my efforts to improve 3D printing technologies in three distinct aspects:

• Printability. In Chapter 2, we present an algorithm to determine structural weakness as well as worst-case usage scenarios. We extensively validated our algorithm both numerically and experimentally.

- Microstructure design. In Chapter 3, we explores the possibility of fabricating microscopic frame structures. We describe an entire pipeline for designing spatially varying microstructures that match target material properties or deformation goals.
- Robust mesh processing. In Chapter 4, we proposed an algorithm to perform robust solid geometry operations using mesh arrangements. We validate our algorithm by processing 10,000 "wild" models crawled from Thingiverse, a popular online shape repository. A detailed analysis of our dataset is presented in Chapter 5.

Chapter 2

Worst-case Structrual Analysis

This chapter is based on the publication ([145]) in collaboration with Julian Panetta and Denis Zorin. For this project, I have worked on the development of all steps of the algorithm (linear elasticity system, modal analysis, weak region extraction, worst-cast load optimization, fabrication, extensive material testing, design and execution of the validation experiments) except accelerating pricipal stress computation.



Figure 2.1: Algorithm overview

2.1 Introduction

We present an algorithm approximating the solution of the following problem: From the shape of an object and its material properties, determine the easiest (in terms of minimal applied force) ways to break it or severely deform it.

Our work is largely motivated by applications in 3D printing. The cost of 3D printing has decreased significantly over the past few years, and the industry is undergoing a rapid expansion, making customized manufacturing in an increasingly broad variety of materials available to a broad user base. While many of the users are experienced creators of digital 3D shapes, engineering design expertise is far less common, and widely used 3D modeling tools lack accessible ways to predict the mechanical behavior of a 3D model.

There are a number of reasons why a 3D model cannot be manufactured or is likely to fail:

(1) the dimensions of thin features (walls, cylinder-like features, etc.) are too small for the printing process, resulting in shape fragmentation at the printing stage;

(2) the strength of the shape is not high enough to withstand gravity at one of the stages of the printing process;

(3) the printed shape is likely to be damaged during routine handling during the printing process or shipment;

(4) the shape breaks during routine use.

In most cases, the first problem is addressed by simple geometric rules ([123]), and the second is a straightforward direct simulation problem. Our focus is on the other two problems. On the one hand, many 3D printed objects are manufactured with a specific mechanical role in mind, and full evaluation is possible only if sufficient information on expected loads is available. On the other hand, jewelry, toys, art pieces, various types of clothing, and gadget accessories account for a large fraction of products shipped by 3D printing service providers. These objects are often expected to withstand a variety of poorly defined loads (picking up, accidental bending or dropping, forces during shipping, etc.).

To predict structural soundness of a printed object, we look for *worst-case* loads, within a suitably constrained family, that are most likely to result in damage or undesirable deformations. A direct formulation results in difficult nonlinear and nonconvex optimization problems with PDE constraints. We have developed an approximate method for this search, reducing it to an eigenproblem and a sequence of linear programming problems.

We demonstrate experimentally that our approach predicts the breaking locations and extreme deformations quite well. While primarily designed for 3D printing applications, our method can be applied in any context where loads are unpredictable and structural weaknesses need to be identified.

2.2 Related work

Computational analysis of structural soundness of mechanical parts and buildings is broadly used, but almost always in the context of known sets of loads. While engineers routinely need to evaluate soundness of structures and mechanisms under worst-case scenarios, in most cases, worst-case loads are designed empirically for specific problems (e.g., construction of buildings to withstand loads from flooding or earthquakes). Automatic methods are less common: an important set of methods in the context of modeling under uncertainty is based on the idea of *anti-optimization* (e.g., [48]). Our work is partially inspired by these concepts.

In aerospace engineering, filter-based methods were developed to predict worst-case gusts and turbulence encountered by an airplane. E.g., [141, 50] model the aircraft's response to turbulence as a linear filter's response to Gaussian white noise. From this model, a worst-case noise sample and resulting strain are obtained.

In the context of analysis tailored to 3D printing applications of the type considered in this paper, the closest work to ours is [120]. The paper proposes to evaluate 3D shapes in two main scenarios to discover structure weakness: applying gravity loads and gripping the shape using 2 fingers at locations predicted by a heuristic method. This set of fixed usage scenarios is often insufficient to expose the true structure weakness for many printed shapes, as discussed in greater detail in Section 2.5. The paper also describes methods for automatic improvement of objects. [123] focuses on purely geometric ways to evaluate whether a structure is suitable for 3D printing based on empirical rules formulated by the 3D printing industry ([140], [112]).

Structural stability for simple furniture constructed from rigid planks connected by nails is analyzed at interactive rates in [131]. Their system also suggests corrections when shapes with poor stability are detected.

A number of recent works address various aspects of computational design for 3D printing. [22] provide a pipeline to print objects in a composite material that reproduces desired deformation behavior. To achieve this goal, the authors accurately model the nonlinear stress-strain relationship of their printing materials and how printed models will respond to imposed loads. The space of deformations is a user-supplied input, and structural soundness of the design with respect to other loads is not considered. While some specialized work on CAD for 3D printing exists, (e.g., the system for heterogeneous material design [68]), overwhelmingly, standard tools with little or no analysis support are used.

[79] proposes a framework to decompose 3D shapes into smaller parts that can be assembled without compromising the physical functionality of the shape so that larger objects can be printed using printers with a small working volume. They use a standard finite element simulation to estimate stress of the input shape under gravity in a user specified upright orientation. Other works aim to print articulated models that maintain poses under gravity but do not require manual assembly. [29] designs and fits a generic, parametrized printable joint template based on a ball and socket joint. Their joint provides enough internal friction and strength to hold poses and survive manipulation, but they tune its parameters experimentally instead of using a physically-based optimization. [11] designs a similar ball and socket joint and a hinge joint. An approximate geometric optimization of stresses is performed by maximizing certain cross-sectional areas of the joint.

3D printing has also been used to reproduce appearance: [59] and [44] optimize the layering of base materials in a 3D multi-material printer to print objects whose subsurface scattering best matches an input BSSRDF.

Our method relies on using eigenmodes of the shape. Modal analysis has proven useful in many contexts. The use of Laplacian eigenmodes of simple shapes for computation predates computers [125] and has a long history in model order reduction for a variety of applications including nonlinear elasticity (e.g., [98]). In graphics literature, [102] first introduced eigenmodes as a basis suitable for simulation applications, and more recently, a number of deformation-related algorithms based on eigenmode bases were proposed, e.g., [58, 14, 15].

At the same time, *experimental* modal analysis (applying periodic forces with different frequencies and measuring displacements at various points) is broadly used to detect structural damage [49].

Finally, [104] presents an overview of several simulations and experiments exploring how printing parameters (build orientation, layer thickness, scan path and speed, temperature, etc.) affect the accuracy and strength of simple shapes. The goal of these works is to evaluate and improve printing technology itself rather than detecting or fixing deficiencies in the input shape.

2.2.1 3D printing processes

To motivate the design of our structural analysis process, we briefly review the most commonly used 3D printing processes and the types of structural problems one can expect. The most relevant aspects of 3D printing processes for structural analysis are the mechanical characteristics of materials produced at different stages and typical loads on the object during and after the production process.

Common *single-stage* 3D printing processes either deposit the liquid material only in needed places (e.g., FDM) or deposit material in powder form layer-by-layer and then fuse or harden it at points inside the object (e.g., stereolithography uses photosensistive polymers, and laser sintering fuses regular polymers by heat).

These processes typically use flexible polymers with large elastic and plastic zones in their stress-strain curves. These polymers rarely break if geometric criteria for printability are satisfied, but they can undergo large plastic deformations.

Printing metal, ceramics, and composite materials often involves multiple stages. For example, the object may be printed layer-by-layer in metal powder with polymer binder. At the next stage, the binder is cured in a furnace, resulting in a *green state* part, and at the last stage, the metal is fused in a furnace and extra metal is added. Green state is brittle and has low strength, so parts in this state are easily damaged. A simpler multistage process is used for relatively brittle composite materials, e.g., gypsumbased multicolor materials: a second curing stage is used to give the material additional strength. Both the green state and the final material are relatively brittle. Whenever binding polymer is mixed layer-by-layer with a different material, the resulting material is likely to be highly anisotropic.

To summarize, both brittle and ductile materials are of importance. The former requires predicting where the material is likely to break, and the latter requires predicting extreme deformations likely to become plastic. Due to the layer-by-layer nature of the printing process, anisotropy is common and needs to be taken into account. Some of the loads even during production stages are hard to predict and quantify.

Goals. These considerations lead to several possible structural analysis goals, unified by a common theme of identifying *worst-case* loads in a family of loads satisfying some constraints (bounds on total force, pressure, direction etc.). The worst-case load is understood to be the one that leads to maximal damage, which can be measured by a norm of stress, maximal displacements, and various functions of these quantities.

2.3 Worst-case structural analysis

Next, we present a formal description of the problem. This formulation is computationally intractable, but it is needed as a foundation for a practical approximate version described in Section 2.4.

Linear elasticity. We use an anisotropic linear material model and the linear elasticity equations to model object behavior for the purposes of determining weak spots and worst-case force distributions. This model is adequate for some materials used in 3D printing, but nonlinear models may be necessary for others, as discussed in greater detail in Section 2.6. We emphasize that a distinction should be made between simulation with given loads used to determine precise stress distributions and computation used to determine approximate worst-case loads: lower accuracy is acceptable for the latter. We briefly review the standard elasticity equations to introduce notation. The stress-strain relationship is linear, and stress is related linearly to displacement:

$$\sigma = C : \epsilon \quad \epsilon = \frac{1}{2} \left(\nabla u + \nabla u^T \right), \qquad (2.1)$$

where ϵ is the strain tensor, σ is the stress tensor, and u is the displacement. C is the rank-4 elasticity tensor, C_{ijlm} , and the notation $C : \epsilon$ denotes application of this tensor to the strain tensor ϵ , $\sum_{l,m} C_{ijlm} \epsilon_{lm}$. We discuss the choice and effects of elasticity tensor C in greater detail in Section 2.6. We assume an orthotropic material, for which the tensor C_{ijlm} has up to 9 independent parameters. In a coordinate system aligned with the material axes, if we represent C as a 6×6 matrix acting on vectors of components of the symmetric strain tensors $[\epsilon_{11}, \epsilon_{11}, \epsilon_{33}, 2\epsilon_{23}, 2\epsilon_{31}, 2\epsilon_{12}]$, its inverse is given by

$\frac{1}{Y_1}$	$-\frac{\nu_{21}}{Y_2}$	$-\frac{\nu_{31}}{Y_3}$	0	0	0
$-\frac{\nu_1}{Y}$	$\frac{1}{1}$ $\frac{1}{Y_2}$	$-\frac{\nu_{32}}{Y_3}$	0	0	0
$-\frac{\nu_1}{Y}$	$\frac{3}{1} - \frac{\nu_{23}}{Y_2}$	$\frac{1}{Y_3}$	0	0	0
0	0	0	$1/G_{23}$	0	0
0	0	0	0	$1/G_{31}$	0
0	0	0	0	0	$1/G_{12}$

where Y_i are directional Young's moduli, G_{ij} are shear moduli, and ν_{ij} are Poisson ratios for different pairs of directions, satisfying $\nu_{ij}/Y_i = \nu_{ji}/Y_j$.

For dynamic linear problems with volume force density F, the equation of motion is

$$\nabla \cdot \sigma = F + \rho \ddot{u},\tag{2.2}$$

where ρ is the density, and the dot signifies the time derivative. We are primarily interested in static problems, but as we use modal analysis at an intermediate stage, we retain the term $\rho \ddot{u}$.

Equation 2.2 is subject to boundary conditions: we primarily use a surface force density F_S , which is captured by the condition $\sigma n = -F_S$ on the boundary of the object. If the object is attached to a rigid support, Dirichlet conditions u = 0 can be imposed on a part of the boundary.

If the equation of motion (2.2) is written directly in terms of displacement u, we get

$$\nabla \cdot \left(C : \frac{1}{2} (\nabla u + \nabla u^T) \right) := Lu = F + \rho \ddot{u}.$$
(2.3)

Rigid motion, torque and translation constraints for static problems. If the object is not fixed at least at 3 non-collinear points, an arbitrary force distribution will result in motion of the whole object. As we are interested in considering unknown forces with no assumptions on attachment, we need to be able to eliminate global motion. We

achieve this by imposing zero total force and zero total torque constraints, which can be written as

$$\int_{\Omega} F dV + \int_{\partial \Omega} F_S dA = 0,$$

$$\int_{\Omega} F \times (x - x_c) dV + \int_{\partial \Omega} F_S \times (x - x_c) dA = 0.$$
(2.4)

Displacements enter into this system only in the form Lu, and the operator L has infinitesimal rigid motions in its nullspace. To have a unique solution in u, we impose a zero rigid motion constraint, similar to total torque and force constraints:

$$\int_{\Omega} u dV = 0, \quad \int_{\Omega} u \times (x - x_c) dV = 0.$$
(2.5)

Surface force model. In cases of interest, the only volume force is gravity. In all but most extreme cases, gravity does not have a major effect, so we concentrate on surface forces. We restrict the forces in three ways.

- Only positive normal forces allowed: $F_S = -pn$, where n is the surface normal, and p is pressure, thus ignoring friction. This is an important assumption, as for most situations described in Section 2.2.1, friction forces are likely to be significantly lower than normal forces. At the same time, it is hard to model the bounds on ratios between normal and tangential components accurately in the absence of detailed knowledge of loads and surfaces. Without such bounds, any optimization is likely to produce unrealistic tangential results. Similarly, negative surface forces (e.g., electrostatic attraction), are not likely to play a major role and are excluded.
- Pointwise pressure is bounded: $p < p_{max}$. If a pressure may be unbounded, an arbitrarily high stress may be produced at a point on the surface. While highly concentrated forces are possible, these are rare, and we assume that a realistic bound on surface pressure is available.

• *The total force is fixed.* Again, by increasing the total force, arbitrarily high stresses can be obtained.

For example, if our primary target is simulating manual handling situations, one can bound the force by a typical force a human can apply by squeezing, and the maximal pressure is derived from the size of the finger tips.

Problem formulation. It remains to specify the objective function. One commonly used measure of interest is maximal principal stress, $\max_{\Omega} \max_{i=1,2,3} |\sigma_i|$, where σ_i are the eigenvalues of the stress tensor. The complete problem of finding the worst-case force distribution satisfying the constraints of our model and optimizing this objective function, has the form

$$\max_{\Omega} \max_{i=1,2,3} |\sigma_i| \to \max;$$

$$Lu = 0 \text{ on } \Omega, \ C : (\nabla u + \nabla u^T)n = pn \text{ on } \partial\Omega,$$

$$\int_{\partial\Omega} pn dA = 0, \ \int_{\partial\Omega} pn \times (x - x_c) dA = 0,$$

$$\int_{\Omega} u dV = 0, \ \int_{\Omega} u \times (x - x_c) dV = 0,$$

$$0 \le p \le p_{max} \text{ on } \partial\Omega, \ \int_{\partial\Omega} p dA = F_{tot}.$$
(2.6)

Maximal principle stress is a suitable measure if we are interested in failure of materials, which occurs when the stress in a direction exceeds a bound. For plastic transition, the norm or some other function of the *deviatoric stress*, $\sigma - \frac{1}{3} \text{tr} \sigma I$, may be of interest.

We make an interesting observation when the material is isotropic and C can be written as $Y\hat{C}$, where Y is the Young's modulus, and \hat{C} is nondimensional, depending only on the Poisson ratio. Then maximal stress does not depend on Y but only on the Poisson ratio.

Solving this problem yields the worst-case principal stress and, importantly, the pressure

distribution on the surface resulting in this stress. The maximal stress makes it possible to evaluate the likelihood of damage during the production process, shipping or use. Examining the pressure distribution makes it possible to evaluate how likely such loads would be and determine how the structure of the object can be strengthened.

We observe that all constraints in this problem are linear equality and inequality constraints, i.e., the constraints are *convex*. At the same time, the functional is highly non-linear (in fact, not smooth) and non-convex. Replacing maximal principal stress with another point measure maximized over the surface does not change the nature of the problem.

A brute-force solution can be obtained by solving a sequence of problems in which the objective functional max $|\sigma_i|^2$ is maximized for every point and then taking the maximum of all results. Because we are interested in *maximizing* the norm, even these simpler perpoint problems remain nonconvex and nonlinear.

We conclude that solving the optimization problem in general form is impractical, and due to non-linearities and non-convexity, any optimization is likely to get stuck in local minima.

Extension to displacements. An obvious extension of the algorithm is optimizing for maximal displacements. The main change is replacing σ with u in the functional: $\max_{\Omega} |u| \to \max$. This formulation is more relevant for flexible materials.

2.4 Efficient approximate algorithm

Overview. Figure 1 shows the main components of the efficient approximate algorithm for solving (2.6).

There are two problems we need to address to make the solution of (2.6) practical: (1) the need to solve an optimization problem for each point of the object to determine
which one results in minimal stress; and (2) the nonlinearity and nonconvexity of each subproblem.

To address the first problem, we use a modal-analysis based heuristic that we found to work remarkably well. The second problem is solved by using tr σ as the *linear* objective functional. The reasons this substitution is possible for a broad range of cases are discussed in detail below.

Modal analysis and weak regions. A crucial ingredient of our method is *modal* analysis, which we use to restrict the part of the object where we need to maximize the stress or another functional.

Computational modal analysis refers to computing eigenvectors (eigenmodes) \mathbf{u}_i and eigenvalues λ_i of L:

$$Lu_i = \lambda_i u_i, \ i = 1, 2 \dots \tag{2.7}$$

It is widely used in engineering and graphics for a variety of purposes. In the context of structural analysis, the most common application of modal analysis is to predict possible damage or deformations in presence of vibrations.

Our idea is similar in spirit, however there is a significant difference. We do not consider vibrations, i.e., periodically changing loads; rather, we consider static or quasi-static loads. We make the following

Assumption 1: Examining a small number of eigenmodes allows us to find all regions of an object where the stress may be high under arbitrary deformation. While this observation is difficult to prove mathematically, physical intuition suggests that vibrations of an object at different frequencies will result in high stress in all structurally weak regions of the object. Weak regions are those where high maximal stress can be obtained with low energy density relative to other parts of the object. To validate this assumption, we have performed a brute-force optimization on a number of models (Figure 2.7) and compared with the results obtained using weak regions only. We obtain a remarkably good agreement in all cases.

We search for locations of potentially high stress by computing a number, M_m , of eigenmodes and considering a fraction $1 - \epsilon$ of points with highest stress under these deformations.

We define weak regions to be the connected components of this set. Each mode has multiple weak regions, typically associated with local stress maxima. For each mode we select M_r weak regions.

Approximate convex problem. The second important change to the problem is to replace the functional in (2.6) with a functional that can be optimized efficiently and that is minimized by a similar pressure distribution, p, to the original. We focus on the maximal stress, although a similar approach can be used for other functionals. We observe that almost invariably for any deformation and any compressible material with Poisson ratio ν sufficiently different from 1/2:

For points where a principal stress is maximal, other principal stresses are small relative to the principal stress.

We have performed a validation of this observation by running simulations with a variety of loads and computing the ratio of the maximal principal stress to $|\text{tr }\sigma|$. Over 36 models tested, the average ratio is 0.96 with standard deviation 0.25. Figure 2.2 illustrate that the distributions of trace and maximal principle stress are visually similar.

We observe that when this is true, the difference between $|\sigma_{max}| = max_{i=1,2,3}|\sigma_i|$ and $|\sum_{i=1}^{3} \sigma_i|$, i.e., $|\operatorname{tr} \sigma|$ is small, and we can approximate the maximal principal stress with the absolute value of the trace.

As weak regions correspond to the highest stress area, and estimated stress tends to



Figure 2.2: The top 10% volume of largest principal stress (left) and largest trace (right) are visually similar

have a significantly lower accuracy vs. displacement, we use a weighted average of the stress over each weak region. The choice of weighting, as long as it falls off towards the boundary of the region, has relatively small effect on the result. We choose the L_2 norm of the stress computed from the eigenmode as the weight w for averaging the stress trace over each weak region. We also predict whether each point will stretch or compress under the worst-case load by computing tr σ under the modal displacement. We choose w's sign to match this quantity.

We finally arrive at the following approximate problem formulation:

For each eigenmode $i, i = 1...M_m$ and each of its weak regions, $D_{ij}, j = 1...M_r$, we solve the following linear programming problem:

$$\int w \operatorname{tr} \sigma dV \to \max \text{ w.r.t. } u \text{ and } p;$$

$$Lu = 0 \text{ on } \Omega, C : (\nabla u + \nabla u^T)n = pn \text{ on } \partial\Omega,$$

$$\int_{\partial\Omega} pn dA = 0, \int_{\partial\Omega} pn \times (x - x_c) dA = 0,$$

$$0 \le p \le p_{max} \text{ on } \partial\Omega, \int_{\partial\Omega} p dA = F_{tot}.$$
(2.8)

Unlike the original problem, this problem has a *unique* solution that can be computed efficiently using a convex solver.

Discretization and additional optimizations. We discretize the problem in the simplest conventional way, using piecewise-linear finite elements. The downside of this approach is that a suitable tetrahedral mesh needs to be generated for each input. For 3D printed models, the task is somewhat simplified: as the cost of printing is dominated by the amount of material used, almost all objects printed in practice are effectively thick shells to the extent this is allowed by the structural requirements. For this reason, tet meshing does not increase the number of vertices used to represent the object as much as one would expect.

Let n be the number of vertices, $n_b \leq n$ be the number of boundary vertices, and m be the number of elements. The discretized quantities are: **p** the vector of pressures defined at boundary vertices of dimension n_b ; and **u**, the vector of displacements of dimension 3n.

In discrete formulation, we optimize the functional

$$\mathbf{w}^T V D B \mathbf{u}.$$
 (2.9)

In this formula, V is a $6m \times 6m$ matrix, with the volume of element j repeated 6 times on the diagonal for the 6 components of the stress tensor. D is a $6m \times 6m$ block-diagonal matrix. For each element, the corresponding 6×6 block is the rank-4 tensor C in matrix form. B is a $6m \times 3n$ applying the FEM discretization of $\nabla + \nabla^T$. Finally, \mathbf{w}^T is a vector that computes and weights the stress tensor traces, so that $\mathbf{w}^T V \mathbf{x}$ discretizes $\int_{\Omega} w \operatorname{tr} x dV$.

The discretized static elasticity equation combined with boundary conditions takes the form

$$-K\mathbf{u} + NA\mathbf{p} = 0, \tag{2.10}$$

where K is the standard FEM $3n \times 3n$ stiffness matrix, $K = B^T V D B$. The matrix N is a $3n \times n_b$ matrix of components of surface normals, returning per-vertex components of external forces (0 for internal vertices, pn for the boundary), and matrix A is the $n_b \times n_b$ diagonal vertex area matrix.

The discretized formulation of the total force and torque constraints are:

$$\Sigma N A \mathbf{p} = 0, \ \Sigma T N A \mathbf{p} = 0, \tag{2.11}$$

where Σ is the 3 × 3*n* matrix, summing *n* 3D vectors concatenated into a 3*n* vector, and *T* is $3n \times 3n$ block-diagonal matrix computing the torques of the surface force vectors.

Putting all these together, the discretized optimization problem is:

$$\mathbf{w} \cdot (VDB\mathbf{u}) \to \max \text{ w.r.t. } \mathbf{u} \text{ and } \mathbf{p};$$

$$-K\mathbf{u} + NA\mathbf{p} = 0,$$

$$\Sigma NA\mathbf{p} = 0, \Sigma TNA\mathbf{p} = 0,$$

$$\Sigma_v \mathbf{u} = 0, \Sigma_v T_v \mathbf{u} = 0,$$

$$0 \le p_i \le p_{max} \text{ for all } i,$$

$$\Sigma_s A\mathbf{p} = F_{tot},$$
(2.12)

where Σ_s sums scalars on the surface, Σ_v sums vectors in the volume Ω , and T_v computes torsion for each point. The total dimension of the problem is $n_b + 3n$.

Eliminating displacements. As most of the degrees of freedom in the system are displacements, but the quantities of interest are pressures \mathbf{p} , eliminating \mathbf{u} results in significant speedups (\mathbf{u} can be eliminated even for the displacement maximization problem). The elasticity equation $-K\mathbf{u} + NA\mathbf{p} = 0$ is not sufficient for this; it has a nullspace of dimension 6 corresponding to the rigid motion degrees of freedom, so we need to consider the constraints for zero total rigid motion, $R\mathbf{u} = 0$, where $R = \begin{bmatrix} \Sigma_v \\ \Sigma_v T \end{bmatrix}$. Rewriting

this system in the standard constrained system form,

$$\underbrace{\begin{bmatrix} K & R^T \\ R & 0 \end{bmatrix}}_{C^*} \begin{bmatrix} \mathbf{u} \\ \lambda \end{bmatrix} = \begin{bmatrix} NA\mathbf{p} \\ 0 \end{bmatrix}, \qquad (2.13)$$

where λ is the Lagrange multiplier for the constraint Ru = 0. It is clear from physical considerations that this system is invertible. Let S be the selection matrix $\begin{bmatrix} I_{3n\times 3n} \\ 0 \end{bmatrix}$. Then, we can express \mathbf{u} as $\mathbf{u} = S^T C^{*-1} SNA\mathbf{p}$. In this form, the objective of (2.12) becomes

$$\mathbf{w} \cdot VDB\mathbf{u} = \underbrace{\mathbf{w}^T VDBS^T C^{*-1}SNA}_{\mathbf{f}^T} \mathbf{p} = \mathbf{f}^T \mathbf{p}$$

The displacement-free optimization problem is

$$\mathbf{f}^T \mathbf{p} \to \max \text{ w.r.t. } \mathbf{p},$$

$$\Sigma N A \mathbf{p} = 0, \ \Sigma T N A \mathbf{p} = 0,$$

$$0 \le p_i \le p_{max} \text{ for all } i,$$

$$\Sigma_s A \mathbf{p} = F_{tot}.$$

(2.14)

While the final system has only sparse constraint matrices, it may appear that computing \mathbf{f}^T for the objective functional requires inverting C^* ; we observe however that $\mathbf{w}^T V DBS^T C^{*-1}SNA = \mathbf{f}^T$ can be rewritten as $\mathbf{f} = (SNA)^T \mathbf{q}$, where \mathbf{q} is the solution of the equation

$$C^{*T}\mathbf{q} = SB^T D^T V^T \mathbf{w}.$$
(2.15)

In other words, it is sufficient to be able to solve a linear system with matrix C^* , and the cost of transforming (2.12) to (2.15) is the cost of a single linear solve.

Finally, for modal analysis, we have observed that the results for isotropic models in

particular are well-approximated by simpler eigenanalysis of the Laplacian, which yields a considerable speedup (compare the bottom two rows of Figure 2.7).



Weakest Region's Mode Index

Figure 2.3: Histogram of the mode number (horizontal axis) in which the weakest region appears for the first time.



Figure 2.4: Histogram of the rank of the weakest region in the weak region list sorted by decreasing energy.

Algorithm summary and parameters. The main steps of our approach are

- 1. Compute a tetrahedral mesh Ω for an input triangle mesh.
- 2. Compute M_m modes using an eigensolver.
- 3. For each mode, find M_r weak regions with highest total energy.
- 4. For each weak region, solve the problem (2.14) to obtain worst-case pressure candidate p_i .

- 5. Solve Lu = 0, with boundary pressures specified by p_i , to obtain displacements u_i , and compute actual maximal principal stress σ_i^{max} for each weak region.
- 6. Maximal stress is determined as maximum of σ_i^{max} .

Tetrahedral meshes are generated using tetgen ([115]).

We use MOSEK ([89]) to solve the linear programming problem, UMFPACK ([43]) for linear solves, and ARPACK ([70]) for computing eigenvectors and eigenvalues.

The parameters of the algorithm include M_m , M_r , the choice of threshold $1 - \epsilon$ for weak regions, and a user-defined maximal pressure p^{max} (the latter can be regarded as a part of the definition of the force model).

To determine reasonable choices of M_m and M_r , we have run modal analysis for a large number of modes (150) and a large number of weak regions per mode for a collection of objects. For each object and each mode, we found the weakest region and checked in which mode it first appears. We also computed its rank in the list of that mode's weak regions sorted by decreasing energy. The results (Figure 2.3 and Figure 2.4) indicate that 15 modes and 5 weak regions per mode are sufficient in over 80% of cases.

We use $\epsilon = 0.025$ in all cases; the dependence of the size of weak regions for one mode on ϵ is shown in Figure 2.5.

Figure 2.6 shows two final results of the algorithm. Red arrows are total forces obtained by summing nearby per-vertex force values (pressures are typically concentrated in small areas). Colormaps on the deformed surfaces show weakness maps.

2.5 Validation

We performed validation of our algorithm in several computational and experimental ways.



Figure 2.5: Weak regions extracted from three modes with weakness level cutoff, $\epsilon = .10, .05, .03, .01$.



Figure 2.6: Optimal force vectors and weakest regions on the left, resulting deformations and stresses on the right. The gray images in the background show the undeformed state.

Comparison with direct search for the weakest region. Instead of using the modal analysis stage to identify weak regions and using averaged stress or displacement over weak regions as the target quantity to optimize, we can run the same optimization process directly, treating each tetrahedron as a potential weak region.

We define the *weakness* map as a scalar field on the surface mapping each point to the maximal principal stress at this point obtained by approximate optimization. Using our method yields a partial weakness map on the union of all weakness regions we consider. Figure 2.7 shows a comparison of a complete weakness map, computed using the brute-force approach, with the weakness map obtained by our method. We observe a close agreement between these for all examples in areas where the partial map is defined and never observe high stress values elsewhere.

Dependence on tetrahedral mesh resolution. To keep the cost of computation low, especially in the context of interactive applications or processing large number of

# Tets	Brute Force (s)	Weak Region (s)	Speedup
2723	681.367	1.089	$625.939 \ {\rm x}$
2869	793.362	1.087	$729.907 \ {\rm x}$
2904	894.610	0.641	$1396.071 \ {\rm x}$
5332	2120.361	1.171	$1810.199 \ {\rm x}$
11020	11029.721	2.729	$4042.403 {\rm ~x}$
12853	11334.362	1.694	$6692.546 \ {\rm x}$
14163	27775.900	3.373	$8234.925 \ {\rm x}$
16008	19917.838	1.892	$10527.388 \mathrm{x}$

Table 2.1: Stress analysis timings for brute force optimization vs. weak region optimization. While speedups are already dramatic for extremely small element counts, the higher asymptotic complexity of brute force causes a rapidly increasing speedup for larger models.



Figure 2.7: Comparison of the similar optimum stresses found by brute force, Laplacianbased weak region analysis, and stiffness-based weak region analysis. The table reports 99.75 percentile (by volume) element stresses. An isotropic metal material was used for this comparison.

objects at a printing facility, using coarse tetrahedral meshes is desirable. As Figure 2.8 shows, weakness maps for different resolutions are similar, so higher resolution may be used only at the last stage, after the weakest spots are identified.



Figure 2.8: For 5 different mesh resolutions (from left to right, the vertex counts are 5K, 13K, 24K, 36K, 50K), the algorithm generates consistent weakness maps.

Drop test. To verify our method for brittle materials, we performed a randomized deformation test by dropping printed models onto horizontal pegs. We dropped the models from 1m high, ensuring a nearly random impact orientation and force application. The test setup is pictured in Figure 2.9. All models were printed with material zp150.

Specific breakages may have two origins: high point forces, which can break even relatively strong spots near the impact point and will vary across drops, and smooth deformations, which are likely to break weak regions consistently. The former does not correspond to typical usage scenarios, which feature distributed bounded forces. Thus, we consider only fractures occurring frequently across drops.

The test results, displayed in Figure 2.10, confirm that the weak regions determined by our method generally agree with the areas with highest occurrence of fracture. Notice in particular the legs of the cow (3^{rd} row, left), the notches of the gear (5^{th} row, left), the arms of the dancer (1^{st} row, right), and the inner piece of the powercog pendant (6^{th} row, left). These are all regions of high weakness map value that break consistently.



Figure 2.9: We used models printed in green state "sandstone" for the drop tests. The testing models often are covered with a loose layer of powder that shakes off upon impact (see the dust in the right image).

Displacement test. For the objects printed in ductile materials, we performed a different test. We placed the shapes into a cardboard box filled with packaging material and applied pressure to the box's exterior. This pressure permanently deformed the models inside. We took photographs of the deformed models in a registered position and compared them to the 3D model from which they were printed. We observe good agreement with the computed map of maximal displacements, i.e., the map similar to the weakness map, but for the displacement maximization problem (see Figure 2.11).

Comparison with [120]. We compare to the approach described in [120], as they also aim to predict the loads that a printed model is likely to experience. The authors use a more specific force model: pinch grips. They present an empirical model to predict how the object will be gripped with two fingers. There are many designs for which such a grip does not capture typical use cases or mishaps. Figure 2.12 demonstrates shapes whose worst-case loads cannot be applied or approximated using only a pinch grip.

Figure 2.13 shows three examples for which the authors of [120] have provided us their force application points. Their "cup" example (left) is an excellent candidate for the pinch grip; the highest stress achieved with a fixed total force agrees with ours and even exceeds it. However, the other two objects do not fit their model as well. The "UFO" pinch grip is clearly suboptimal, and the forces applied to the bracelet would have much more leverage if they were moved to the open endpoints. In all three cases, our method



Figure 2.10: Results for a drop test. Model volume is shaded with its weakness map percentile: 90% 99%



Figure 2.11: Simulation results (left) are compared to the deformed 3D printed model (green) overlaid on an undeformed one (blue). Our algorithm predicts likely regions (red) of large deformation under normal handling. For the blade earring, we confirm that the largest blade deforms little relative to the hook and shaft: after aligning the shafts to be parallel, the largest blades are also roughly parallel (see the yellow parallelograms). The second largest blade is displaced more. Note that the upper right pin of the deformed spinnoloid (middle row, green) was broken during printing.

generates efficient force vectors.

An interesting observation about the "cup" model is that our method produces a triangle of forces (perhaps at the expense of higher stress) rather than a pair of opposite forces. One possible reason for this is the pressure bound requiring the force to be distributed over a larger area.

Timings. Though our pipeline has not yet reached interactive speeds, it is already fast enough to be included in a 3D printing pipeline. For the sizes of models most commonly sent to 3D printing services (see distribution in Figure 2.15: sizes on the order of 100K



Figure 2.12: Models where pinch grips cannot generate worst-case loads. Our method finds highly intuitive force vectors, regardless. The additional arrows on the top of the Skyrim dragon arise to bring the total force and torque to zero.



Figure 2.13: Comparison against [120]. Our algorithm's force distribution (top) better identifies structural weakness, especially for the ufo (middle) and bracelet (right).

vertices are most common), our full algorithm takes only a few minutes:

# Tets	Structural Analysis (mins)
2723	0.028
42900	0.308
70356	0.382
155383	2.566
322398	9.601
414894	4.490

Analyzing the algorithm's scaling behavior is complicated by its dependence on structural properties—a separate linear program is run for each weak region that is extracted. To make sense of the timings, they have been separated by stage. Modal analysis and weak region extraction are run only once per model, and Figure 2.14 shows how their execution times depend on element count. The time spent setting up and solving the linear programs ("weakness analysis") is averaged over all weak regions so that it can be plotted against the same x axis. Note that there is one further cost not shown: the single sparse UMFPACK factorization. This timing depends strongly on matrix structure (despite using fill-in reducing permutations), and adds noise to curves when included. Factorization time is included in the timing table above.



Weak Region Computation Time

Figure 2.14: The top plots shows how modal analysis and weak region extraction scale with the number of tetrahedra. The dominant cost is the eigensolve. The bottom plots shows the average cost of setting up and running the linear program for each weak region. It excludes the UMFPACK factorization of C^* that only must be run once.



Figure 2.15: Model vertex counts tabulated from 2781 models ordered from Shapeways.

2.6 Material properties

Material parameters defining the elasticity tensor C must be measured for each of the 3D printers' materials. We have observed that the computed maximal stress does not depend on the magnitude of the Young's modulus in the isotropic case. However, in the anisotropic case, it does depend on the ratios of directional elasticity moduli, which can be significant (Figure 2.16). To predict breakage or plastic deformations under loads, the additional material parameters tensile strength and yield strength are needed.

In this section, we present the Youngs modulus ratio measurements for three different 3D printing materials that we used to compute our simulation's elasticity parameters. In addition, we discuss the extent to which various materials match our assumptions on stress-strain linearity and what accuracy one can expect from predictions of the maximal stress to tensile strength ratio. In all cases, we assume a Poisson ratio of 0.3.

We have tested three materials used in 3D printing: nylon (PA 2200 by EOS Electro Optical Systems), "sandstone" (zp150 used in the ZPrinter series by 3D Systems), and green state stainless steel (420SS powder bound with proprietary binder used by Ex-One). They also represent different classes of materials (brittle vs. ductile, isotropic vs anisotropic).

To determine their properties, we conducted a three point bending test consistent with



Figure 2.16: Different ratios of directional Young's moduli can lead to different weakest regions. We show the weakest region found for a truss with a Young's modulus that is five times higher in the X (left), Y (middle), and Z (right) directions.

ASTM standard D5032 ([7]), using the Instron 5960 universal testing machine with a ± 100 N load cell and a support span of 40mm. Figure 2.17 illustrates the testing setup. The testing samples are rectangular bars with length 60mm and thickness between 1mm and 5mm. A relatively thin test bar was chosen because structurally weak models are likely to contain thin features.



Figure 2.17: Three-point bending test on green state stainless steel.

Among the three materials tested, green state stainless steel fits the definition of brittle material the best. Stress grows linearly with strain for all samples tested (Figure 2.18 left). Bending tests in perpendicular directions show that elastic moduli in these directions are close, with the average Young's modulus 3.59GPa and standard deviation 0.27GPa. Figure 2.19 shows critical stress extracted from measurements, which is mostly consistent for all samples, with the average 6.88MPa and 0.62MPa standard deviation.



Figure 2.18: Left: Stress vs strain curve measured on samples in green state stainless steel. The colors indicate different sample thickness (1.5mm red, 2mm green, 3mm blue). Right: Stress vs strain plots for nylon testing samples of thickness 1.5mm and 2mm. The samples printed in different orientation are marked with different colors (red: X, green: Y, blue: Z).

Overall, this material is consistent with our model for stress optimization.



Figure 2.19: The critical stress distribution of green state metal for samples with thickness 1.5mm up to 5mm.

Models printed in nylon are known to withstand a large range of deformations. Figure 2.18 (right) shows the stress vs strain curve for 18 nylon samples. Half of them are 1.5mm thick, and the other half are 2mm. For each thickness group, sets of 3 samples were printed along each of X,Y and Z directions. From the results, we observed that nylon samples typically have a very large elastic deformation range before entering the plastic stage. We also note a moderate but obviously present degree of anisotropy (the Young's modulus for X is 0.80GPa with 0.13GPa deviation, for Y is 1.02GPa with 0.18GPa deviation, and for Z is 0.98GPa with 0.12GPa deviation). See Figure 2.18, right.

The most complex material we tested is the "sandstone." Though, like green state metal,

it has a relatively low tensile strength, it exhibits a significant plastic region (Figure 2.20) and very high degree of anisotropy: we measured X, Y, and Z Young's moduli of 1.22GPa (standard deviation 0.13GPa), 0.68GPa (standard deviation 0.07GPa), and 0.234GPa (standard deviation 0.02GPa) respectively, with more than 5 times difference between the largest and smallest values. Thus, we model it as an orthotropic material with a distinct Young's modulus per printing axis. We obtain our shear moduli using a standard formula from [119]: $G_{xy} = \frac{E_x E_y}{E_x + E_y + 2E_y \nu_{xy}}$. Note that "sandstone" exhibits a large variability of tensile strength, even for a single direction. This means only very conservative predictions are possible. Nevertheless, we observe that our weak region detection works well (Figure 2.10).



Figure 2.20: Stress vs. strain measurements on rectangular bars printed with green state "sandstone" along the printer X (red),Y (green) and Z (blue) direction. Different printing directions influence the material properties significantly.

2.7 Conclusions

We have presented an efficient approximate method for determining worst-case loads for a geometric object based on its geometry and material properties only. The method is quite reliable (it relies on a linear solver, an eigensolver, and a convex solver, which all can provide convergence guarantees), efficient, and approximates well the worst-case stress and displacement distributions.

At the same time, there is clearly a number of limitations. Most importantly, only linear elasticity is considered, and the optimized solution at the second stage may not match reality for large plastic deformations. We note, however, that the robustly obtained approximate solution can serve as a starting point for a nonlinear solver. More generally, 3D printed materials exhibit a broad range of complex behaviors, some of which may exhibit considerable variation even for the same printing process. Using computational models reflecting material complexity and uncertainty is an important future direction. From the point of view of robustness of the method, tetrahedral mesh generation is the bottleneck. Meshless techniques may yield a fully robust pipeline using only surface geometry as input.

Chapter 3

Elastic Textures for Additive Fabrication

This chapter is based on our publication [100] in collaboration with Julian Panetta, Luigi Malomo, Nico Pietroni, Paolo Cignoni, Denis Zorin. My contributions include the development of 3D wire inflation system based on [57] and extend it to support periodic inflation, grid-based and mesh-based tiling of mixed single-cell patterns; research and testing of 3D printers' capability of fabricating microstructures; formulation of selfsupporting and tilable constraints; initial implementation of linear elasticity system with heterogeneous material support; initial implementation of parameter sweep; proposal and initial implementation of local-global material optimization algorithm; implementation of material properties to single cell pattern lookup algorithm using norm of compliance tensor difference; testing of the raw material properties; fabrication and testing of all examples used in the paper.



Figure 3.1: Six basic elastic textures are used to obtain a large range of homogenized isotropic material properties. A $3 \times 3 \times 1$ tiling of each pattern is shown, along with rendered (left) and fabricated (right) cell geometry below. The naming convention is explained in Section 3.4.

3.1 Introduction

Rapid advances in the accessibility of additive fabrication has a significant impact on how manufacturable geometric models are constructed. A key distinctive feature of common additive fabrication technologies is that the cost and time of production is practically uncorrelated with structural complexity: in fact, a complex structure using less material may be both cheaper and faster to produce.

Complex structures, aside from potentially reducing costs, open up many new possibilities, in particular for manufacturing deformable objects. By varying a small-scale structure, one can adjust a variety of material properties, from elasticity to permeability. Importantly, these properties can be varied nearly continuously over the object, something that is not commonly done in traditional processes. As it was observed in prior work, this opens up many new possibilities for object behavior.

Small-scale structures present a set of new design challenges: in all but the simplest cases, these are hard or impossible to design by hand to meet specific goals. At the same time, computational optimization of fine-scale variable structure over a whole object, even of moderate size, can easily result in numerically difficult topology and shape optimization problems with millions of variables.

In this paper, we describe *elastic volumetric textures*, a library of tileable parameterized

3D small-scale structures that can be used to control the elastic material properties of an object. Applying such textures to a hex mesh with target material properties specified per element is similar to using dithering to achieve a continuous variation of brightness or color.

In a sense, almost *all* material properties owe themselves to small-scale structures at the molecular or crystal level, and a large body of work in nanoscience aims to control material properties precisely by structure design. These works must accommodate constraints imposed by the specific properties of the elements and molecules used, the need for self-assembly, and other considerations.

Our focus is on larger-scale structures, which can be manufactured using existing 3D printing technology. With feature sizes at the scale of 10μ m- 100μ m, these are well described by conventional elasticity theory. While this type of structure was also extensively studied, typically this was in the context of a specific problem, such as optimizing strength for a given material volume fraction. Our goal is to maximize the range of effective material properties that can be obtained using a single material by varying the structure.

We consider variable-thickness *truss-like structures*—i.e. structures composed of connected bars—as these cover a considerable range of properties on the one hand, and on the other hand, allow us to work with a relatively small number of parameters. We present a method for building a dictionary of structures that cover a large space of material properties. These structures are *tileable*, which makes it possible to vary material properties across an object, and *printable*.

We demonstrate that elastic volumetric textures allow one to control the deformation behavior of objects, either by painting material properties directly or by a two-stage *shape optimization* procedure, involving solving for variable continuous properties then approximating them using our texture dictionary. We validate our results by measuring samples for different choices of parameters and topologies and by demonstrating the deformation behavior of objects fabricated with spatially varying structures.

3.2 Related Work

Microstructure design and optimization. There is a huge literature on theoretical studies of effective moduli of composites (our periodic structures are an extreme example of a composite combining a material with void). Recent monographs include [42, 85, 126]. Much of the literature focuses on identifying microstructures with *extremal* effective behavior, i.e., with effective elasticity properties at the boundary of the achievable zone for a given class of composites [4, 38, 85]. Many classes of extremal structures were described (see, e.g., [28]), however most of these classes—e.g. sequentially laminated microstructures [10] and microstructures based on inclusions [53, 78]—are either difficult or impossible to manufacture at this time. Interchangeable composites and other structures were found that maximize simultaneously, e.g., the bulk modulus and permeability [56] or electrical conductance [128, 129, 127], but these designs are of limited use for tailoring elastic behavior.

The closest work to ours is [117], which constructs truss microstructures with prescribed elasticity tensors. It starts with a full "ground structure" containing about 2000 candidate members, then optimizes the members' thicknesses *but not offsets* to obtain a microstructure period cell whose homogenized properties (computed using a truss model) match the desired properties. Neither tileability of structures for different parameters nor printability can be guaranteed. We discuss the differences in greater detail at the end of Section 3.4. Further exploration of periodic structures of this type was done more recently in [40], comparing different methods for optimizing these structures.

A number of microstructures were obtained using various types of *topology optimization*, which was originally designed for global structure optimization. In the case of microstructure design, these methods look for a periodic structure minimizing, e.g., compliance for a fixed total volume fraction. The result is normally a single-scale structure, with scale controlled by the resolution of the simulation grid or other types of regularization. Important methods proposed for solving these problems include solid isotropic material with penalization (SIMP) and rational approximation of material properties (RAMP) [17, 18, 95]. [106] demonstrated design of isotropic materials maximizing bulk modulus.

Topology optimization offers more flexibility in the choice of structure, but it requires a relatively expensive optimization for each specific problem. The ability to undergo topological transitions under continuous parameter changes is both a strength, as it allows exploration of a broader space of structures, and a weakness, as it considerably complicates design of parametric families satisfying printability and tileability constraints, which motivates our approach.

Microstructure fabrication. Several groups focusing on additive fabrication have recently obtained encouraging results. In particular, materials previously thought to be unmanufacturable were produced and behave as expected. Notably, the work of Hollister and collaborators [77, 76, 62, 66] in the context of bone scaffold design and fusion cage design demonstrated the use of optimized microstructures. The possibility of manufacturing auxetic (negative Poisson's ratio) materials was demonstrated in [55], and in [111, 26, 5].

The idea of fabricating tileable structures with varying properties also appears in [61] in which the authors discuss "digital materials," as composed of a set of discrete voxels with predefined shapes that can be connected. Similarly, a building-block based approach was also used in the context of bio-printing [86], where the authors use spheroids of living materials with evolving and controllable composition, varying material and biological properties in time.

Compliant mechanisms. The material optimization method that we present solves a similar problem to that of compliant mechanism design. [18] reviews several existing approaches to designing mechanisms that maximize mechanical advantage/output deflection or tune an output displacement to a particular path. These approaches have little control over the resulting structure's macroscopic shape, whereas tuning deformation behavior using our microstructure approach creates a "mechanism" that still looks like the input shape.

Fabrication and computer graphics. A broad variety of fabrication-related work has been done in the computer graphics community. Several techniques have been proposed to design paper craft objects [87], plush objects [88], and objects made of interlocking planar slices [41, 109, 60]. Other techniques use geometric techniques to change surface appearance by synthesizing surface microgeometry [137] or changing the shape to generate custom target caustics [110].

Another close work to ours, [22], introduces an optimization process to find the best combination of stacked layers to satisfy an input deformation, enabling fabrication of objects with complex heterogeneous materials using multi-material 3D printers. Our work can be viewed as complementary, focusing on the design of structures that can be, e.g., used as a part of deformation behavior design; our material optimization method provides an alternative to the method in that paper. In [118], multi-material printing and discrete material optimization is used in a similar way on complex characters to achieve desired deformations with actuation. Our elastic textures can be viewed as a tool for solving this type of problem. Our structures also can be employed in systems like [36] and [135].

Homogenization. A central tool in our work, homogenization was used in graphics for reducing complexity of physical models in [67], finding the constitutive parameters of a low resolution discretization that best approximates the behavior of the original higher complexity material. The periodic homogenization method that we use is based on the one described in [4].

3.3 Overview and Main Results

In this section, we describe our overall approach, visualized in Figure 3.2, and a specific set of patterns that we have obtained.



Figure 3.2: Overview of elastic texture generation and use.

Problem. The general problem we solve can be formulated as follows: for each tensor C from a given range of elasticity tensors, and a base isotropic material with Young's modulus E^b and Poisson's ratio ν^b , find a structure made out of the base material in a unit cubic cell, such that if the cell is infinitely tiled in space, the resulting homogeneous material has elasticity tensor C.

As discussed in the introduction, we aim to construct a family of patterns that are printable and tileable to enable creation of variable material properties.

Printability is heavily dependent on the choice of technology. We focus on printability criteria related to stereolithography, the most accurate 3D printing method available at this time, but our approach can be easily modified to handle other technologies.

As the printing process proceeds layer-by-layer, we assume that the structure is defined with respect to a fixed coordinate system X, Y, and Z aligned with the printer, with Zbeing vertical. The (idealized) printability criteria that we use are:

- 1. There are no enclosed voids.
- 2. For any point of the structure, the extent covered by the structure in the X, Y, and Z directions from the point are above a printability threshold d_{\min} .
- 3. Every point of the pattern is supported: for every XY slice, all connected components of the slice have at least one point connected to lower points in the structure by a segment contained in the structure. While this condition does not prevent long horizontal bars supported at single points, which can be difficult to print, we have found it sufficient in practice for pattern sizes up to 10mm and a d_{\min} of 0.3mm.

We also make our primary goal to generate periodic structures with *isotropic* homogenized properties. Such patterns have the elasticity tensor C defined by two parameters, Young's modulus E and Poisson's ratio ν , and its inverse, compliance tensor S, has the (Voigt notation) form

$$S = \frac{1}{E} \begin{pmatrix} 1 & -\nu & -\nu & 0 & 0 & 0 \\ -\nu & 1 & -\nu & 0 & 0 & 0 \\ -\nu & -\nu & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 2(1+\nu) & 0 & 0 \\ 0 & 0 & 0 & 0 & 2(1+\nu) & 0 \\ 0 & 0 & 0 & 0 & 0 & 2(1+\nu) \end{pmatrix}.$$
 (3.1)

Expressing in terms of the shear modulus, $G = E/(2(1 + \nu))$, the last three diagonal terms of S are simply 1/G.

While for many tasks anisotropic materials are either sufficient or preferable, periodic structures with isotropic homogenized properties are easiest to use, as cell orientation is decoupled from material properties. In addition, once an isotropic starting point is obtained, it is easy to obtain a controlled anisotropic behavior.

Searching the space of all possible structures in a cell, even at a finite resolution, is an impossible task. Instead, we choose a space of structures with a limited but sufficiently large set of parameters, that can be optimized to achieve specific material properties.

Truss-like structures. We focus on truss-like structures (*patterns*) as shown in Figure 3.1, consisting of bars of different thicknesses connecting a set of nodes in the cell. Unlike real truss structures, the connections between bars are not pin joints, and flexural rigidity at the nodes plays a major role.

This particular space of structures is motivated by several considerations. First, the space is known to contain both very stiff and very weak patterns, providing a broad range of behaviors. Second, tileability and printability requirements yield specific geometric conditions, expressed mostly as constraints on the structure's geometry. For example, the requirement of no enclosed voids is automatically satisfied if the frame structure has no

self-intersections; the bound on extents can be obtained by bounding the thickness from below; and the support condition is easily formulated as a constraint on node positions. These conditions are detailed in Section 3.4.

Symmetry considerations, as well as restrictions on the number and placement of nodes, yield a space of patterns parametrized by their set of edges connecting some subset of the 15 candidate nodes we define on a tetrahedron (their *topology*), thicknesses of these edges, and offsets of the nodes from their default positions. This space is still very large, and we explore it using both topology and geometry searches, described in Section 3.4. These rely fundamentally on the homogenization and shape optimization procedures described in Sections 3.5 and 3.6.

Resulting family. The search procedure's final result is shown in Figure 3.3; the six pattern topologies themselves are illustrated in Figure 3.1. The complex boundaries of the (E, ν) regions arise from the multiple types of geometric constraints enforcing printability. Figure 3.4 shows some patterns with topology "(E1,E2)(E1,E4)(E2,E4)."

The family of topologies covers a large range of Young's moduli, with a largest-to-smallest Young's modulus ratio of 1800, and a sizable range of Poisson's ratios, -0.16 to 0.48.

We note that the range of negative Poisson's ratios is somewhat limited, while on the high end, we are able to achieve ratios close to the theoretical maximum. This observation is consistent with [117]: while it is relatively easy to obtain more extreme negative Poisson's ratios for patterns with cubic symmetry but with shear modulus too low for isotropy, the isotropy requirements restrict the range. Printability constraints restrict it further.

Quite remarkably, four out of the six topologies can be transformed into each other by simple operations (single vertex splits, addition of cross-shaped supports connecting some nodes). The other two are also related to each other by a simple transform, but are not related to the first sequence.

We do not claim that the proposed family is in any sense optimal. It is most likely

possible to extend the coverage or to cover the same domain with fewer topologies. However, the presented set is already quite useful for controlling material properties, as the examples of Section 3.7 show.



Figure 3.3: Region of the (E, ν) space covered by the selected set of patterns. Each topology's coverage is shown in a different color.



Figure 3.4: Samples of the (E, ν) space reached by patterns with topology "(E1,E2) (E1,E4) (E2,E4)."

Accuracy. We fabricated eight patterns with different homogenized Young's moduli using the B9Creator SLA printer and tested their stiffness using the BOSE ElectroForce 3200 measurement system. The machine gradually compressed our samples in the Z direction between two compression plates and measured the displacement resulting from the applied force at each step. We used $6 \times 6 \times 2$ tilings of 5mm cells for this test.

We note that our measured force/displacement slopes are roughly proportional to the homogenized Young's moduli (Figure 3.5a), implying that the measurements are consistent with some (unmeasured) base Young's modulus. The curvature seen could be explained partly by friction in the compression testing setup (Figure 3.5b). Another significant source of error is the inaccuracy of our B9Creator, which tends to thicken thin geometries.

We used a lower-accuracy setup to measure Poisson's ratio but still obtained reasonable agreement with homogenization (Figure 3.6). We compressed the microstructures in the Z direction between two lubricated metal blocks and manually measured the expansion/contraction in the X and Y directions. From these displacement measurements, we computed the X, Y, and Z strains and their ratios.



Figure 3.5: Compression test results for eight patterns with varying homogenized Young's moduli $(6 \times 6 \times 2 \text{ tiling of 5mm cells})$. (a) Slopes extracted from the measured force vs. displacement curves along with a best-fit line through the origin. (b) Moduli extracted from simulated compression tests, with and without modeling compression plate friction. Without friction, the simulated test agrees with homogenization perfectly, but friction introduces error.

We also validated the patterns' isotropy by printing a block filled with a tiled pattern that



Figure 3.6: Poisson's ratios measured from $3 \times 3 \times 1$ printed tilings of 10mm cells vs. homogenized properties. The $\nu = -0.67$ sample, outside our family's range, violates isotropy and printability constraints (we added support structure manually for this experiment).

was rotated by 45° around the Z axis and clipped (Figure 3.7). The measured effective Young's moduli in the rotated orientations were in good agreement with the unrotated orientation: a compression test in the X, Y, and Z directions extracted effective Young's moduli of 0.635MPa, 0.6293MPa, and 0.628MPa for the example shown.





Figure 3.7: We extracted a 45° rotated rectangular block from a regularly tiled 10mm cell microstructure to test Young's modulus in non-axis aligned directions.

Base Material. We used estimated base material properties of $E^b = 200$ MPa and $\nu^b = 0.35$ for all results. We estimated E^b using a three point bending test on rectangular bar samples, but the base Young's modulus can also be estimated from the microstructure

compression test.

Using a different base Young's material would not qualitatively change our results, apart from making the patterns uniformly softer or stiffer. The homogenized Young's moduli depend linearly on E^b , and the Poisson's ratios are independent of E^b , so scaling the base modulus simply softens/stiffens every pattern by the same factor. In particular, for the displacement-based material optimization of Section 3.7, fabricating the solution with a different base Young's modulus maintains the same target deformation behavior (although the required force will change).

We note that our patterns are not very sensitive to moderate changes in the base Poisson's ratio. Changing from $\nu^b = 0.35$ to $\nu^b = 0.35 \pm 0.05$ results in a median relative change in Young's modulus of 0.683% (max: 1.88%) and a median absolute change in Poisson's ratio of 0.00229 (max: 0.0129) over all patterns. Since most additive fabrication materials fall within this range, we expect similar results for other printers and materials.

3.4 Search for Efficient Patterns

In this section, we describe the class of patterns that we consider and the main steps of the search method.

Ground class of patterns. The topology of patterns is defined by a set of edges connecting nodes in the cube cell. We generate the geometric variations by adding *offsets* to the node positions and by changing edge *thicknesses*.

Motivated by the isotropy requirement, we constrain our search to patterns with cube symmetries, which are guaranteed to have the same Young's moduli and Poisson's ratios in every axis-aligned direction. That is, the compliance tensor has the form (3.1) except the last three diagonal entries 1/G may not equal $2(1+\nu)/E$. This yields an easy-to-check isotropy measure:

$$A = 2(1+\nu)G/E,$$
 (3.2)

which we use to identify isotropic patterns in our search. The symmetry will also dramatically reduce the space of pattern topologies to a tractable size after a few additional constraints are introduced. Note, however, that cube symmetry is not necessary for isotropy; other isotropic structures exist with, e.g., tet symmetry.

Consider the group of symmetries of a cube O_h , which includes reflections about three symmetry planes orthogonal to the X, Y, Z axes and the six planes orthogonal to the bisector of each pair of axes. By partitioning the cube according to these symmetry planes, we obtain 48 equal tetrahedra as in Figure 3.8a. O_h maps a single one of these tetrahedra to any other, so it is sufficient to define the nodes and edges of the pattern graph—as well as their offsets and thicknesses—on a single tetrahedron.



Figure 3.8: (a) The tetrahedral cube decomposition used to generate 3D patterns; (b) The 15 nodes defined on a tetrahedron together with their degrees of freedom.

We generate the different topological configurations by changing the connectivity between 15 nodes on a tetrahedron (see Figure 3.8a): vertex nodes $\{V_0, V_1, V_2, V_3\}$, edge nodes $\{E_0, E_1, E_2, E_3, E_4, E_5\}$, faces nodes $\{F_0, F_1, F_2, F_3\}$, and a single internal node, T_0 . Configurations are named by their graphs' edge sets (see labels in Figure 3.1). Each node is constrained to stay on its respective simplex to preserve the topology, so vertex nodes are fixed, edge nodes have a single offset, and so on (Figure 3.8b).

Figure 3.9 shows an example topology colored by its node and edge orbits with respect


Figure 3.9: Symmetry orbits are colored with yellow, red and green. Left: vertex symmetry orbits. Right: edge symmetry orbits.

to symmetry group O_h , and Figure 3.10 demonstrates the effects of the node offset and edge thickness parameters.



Figure 3.10: The results of varying the thickness (top) and offset (bottom) parameters of a particular pattern topology.

The space of possible connectivities, even after accounting for symmetries, is far too large to explore completely (on the order of 10^{32} configurations). We enforce the following constraints to reduce the space of patterns:

- Connected: the tiled pattern is a single connected component.
- No coinciding edges: no edge is contained within another. E.g., if graph edge (V_0, V_1) is chosen and E_0 is the midpoint node of the corresponding tet edge, graph edge (V_0, E_0) is forbidden since it overlaps the first for any offset.
- No dangling edges: every node has valence greater than 1.
- Number of edges: at most 3 graph edges per tetrahedron.

• Max node valence: node valences do not exceed 7.

Valences are computed on the graph after periodic tiling of the cube cell. The first two criteria reduce the space to 16221 topologies, and the remaining three to 1205 topologies.



Figure 3.11: Two pattern topologies from each of three different families, shown with the families' *interfaces* (nodes on the cube cell faces).

Printability. For truss-like patterns, printability is affected by two main factors: the pattern graph structure and the edge thicknesses.

The first printability criterion can be defined on the nodes by considering their offset positions. We say that a node n_1 has supporting node n_2 , if these are connected by an edge and n_1 is strictly above n_2 . We say that n_1 and n_2 are at the same level if they have equal Z coordinates. A pattern is printable only if every connected set of nodes at one level has at least one supporting node.

Printability can be tested by a simple algorithm: we first mark as supported all nodes with a supporting node (considering periodicity). Then we propagate the front of supported nodes to neighbors at the same level. When this breadth first search terminates, the pattern is printable if and only if all nodes are marked as supported. The procedure is illustrated in Figure 3.12. We also note that this constraint can be expressed algebraically as a set of inequality constraints on the offset variables, which can be enforced by an optimization solver.

Tileability. The tileability requirement means that all pattern topologies should belong to the same *family*, meaning topologies with the same set of nodes and edges appearing on the faces of the cube cell (Figure 3.11).



Figure 3.12: 2D examples of the printability detection algorithm. Vertices with supporting nodes are marked (green), then a breadth-first search extends the supported vertex front to horizontal neighbors. The remaining unmarked nodes are unsupported (red). Two cases are shown: unprintable (top) and printable (bottom).

Searching the space of topologies. The goal of our search is to identify a family of pattern topologies that covers as much as possible of the (E, ν) space while satisfying the printability and tileability requirements.

The initial space consists of all pattern topologies satisfying the constraints on graph connectivity mentioned previously. We proceed in the following steps:

- 1. Coarse geometry sweep. Geometric variations are generated for each pattern topology by trying thicknesses of 0.3mm and 0.7mm and node offsets corresponding to barycentric coordinates of 0.2, 0.35, 0.5, 0.65, and 0.8 on the associated tetrahedron simplex. The resulting *printable* patterns are meshed, self-intersecting meshes are discarded, and the remaining patterns' effective elasticity parameters are computed using periodic homogenization (Section 3.5).
- 2. Isotropy filtering. We select a subset of the patterns closest to isotropic (we use a heuristic bound of 0.8 < A < 1.2), which we consider promising candidates as starting points for optimizing pattern parameters to precisely match a range of isotropic elasticity tensors.
- 3. Topological family selection. At this point, we have a rough map of the area in

 (E, ν) space covered by our set of patterns. We obtain a rough estimate of each topology's coverage by taking the convex hull associated with its nearly isotropic geometric configurations. We manually pick the single family whose pattern topologies cover the largest region of (E, ν) based on these estimates.

- 4. Selection of a minimal covering set of topologies. For the selected family, we run a finer sweep of offsets and thicknesses, again filtering for printability, to compute a more precise estimate of the boundary of the (E, ν) domain that each pattern topology can cover. Among all topologies in the family, we selected 6 such that the union of their coverage areas contains most of the domain covered by the family.
- 5. Lookup map construction. Finally, using the shape optimization machinery of Section 3.6 and the initial nearly-isotropic points for each of the 6 topologies chosen, we optimize each patterns' parameters to reach a grid of isotropic elasticity tensors evenly spaced in $(\log(E), \nu)$.

Our procedure is similar to [117], but with several key differences. First, [117] uses a simplified truss model, whereas our method directly homogenizes and optimizes the printable geometry. Second, by using a full topology (including all possible edges between nodes in a ground structure) and permitting zero edge thickness in optimization, the work avoids the topology enumeration stage. As a side-effect, it cannot accommodate the lower thickness bounds or support criteria needed for printability. While a mixed-integer formulation like [83] could allow enforcement of d_{min} by introducing separate binary variables to disable members, this would involve a difficult mixed-integer nonlinear programming problem in our tensor-fitting setting. Finally, by introducing offset variables, our novel shape optimization approach enables much finer control of the elasticity tensors as the design is not limited to the discrete node positions of a ground structure.

3.5 From Patterns to Material Properties

Our goal is to find the homogenized elasticity tensor C^H , describing the effective properties of the microstructure when it is fabricated at a small enough scale and periodically repeated to fill the space. This elasticity tensor is almost never the spatial average of elasticity tensors (for example, if a cell is almost completely filled with material but is disconnected from other cells, it has zero Young's modulus). We first define more precisely what a homogenized elasticity tensor is and then explain how to compute it.

Defining the homogenized tensor. Consider heterogeneous object Ω^{ϵ} filled with a periodic microstructure, as shown schematically in 3.13. Parameter ϵ determines the size of cell Y relative to the object Ω^{ϵ} and permits asymptotic analysis as $\epsilon \to 0$.



Figure 3.13: (Schematic) Periodic tiling of a domain Ω with base cell Y having geometry ω and length scale ϵ .

The elastic response of an object under macroscopic external load \vec{f} is governed by the linear elastostatic equation

$$-\nabla \cdot [C:\varepsilon(\vec{u}^{\epsilon})] = \vec{f} \text{ in } \Omega^{\epsilon}, \qquad (3.3)$$

complemented by appropriate boundary conditions. Here, C is the periodically varying elasticity tensor, and $C : \varepsilon$ denotes its double contraction with strain $(C_{ijkl}\varepsilon_{kl})$ to compute stress. Considering a sequence of problems indexed by ϵ and letting \vec{u} denote the limit of \vec{u}^{ϵ} as $\epsilon \to 0$, the homogenized elasticity tensor C^H is defined as the tensor that satisfies

$$-\nabla \cdot [C^H : \varepsilon(\vec{u})] = \vec{f} \text{ in } \Omega, \qquad (3.4)$$

with same boundary conditions. Vectors \vec{u} and \vec{u}^{ϵ} denote the displacement, and $\varepsilon(\vec{u}) := \frac{1}{2}(\nabla \vec{u} + (\nabla \vec{u})^T)$ is the Cauchy strain tensor.

Expressions for the homogenized tensor. The standard derivation of the homogenized elasticity tensor based on a two-scale asymptotic expansion is provided in the additional material. Here we give an intuitive motivation for the periodic homogenization equations.

 \vec{u}^{ϵ} has a high frequency periodic component that is averaged out to obtain \vec{u} as period $\epsilon \to 0$. So $\vec{u}(\vec{x})$ can be thought of as the average displacement over the infinitesimal base cell Y at point \vec{x} . Likewise, $\varepsilon(\vec{u})$ is the average strain in the cell. For the object to be in equilibrium, (3.4) should represent an average force balance over the cell, meaning C^H : $\varepsilon(\vec{u})$ should be the average stress tensor. That gives the following intuitive interpretation of C^H : it maps the average strain applied at a point to the average stress resulting within the microstructure geometry.

Thus, applying C^H to $\varepsilon(\vec{u})$ is equivalent to simulating the microstructure's deformation under that average strain and averaging its stress. We formulate this simulation inside a single base cell Y by assuming that the displacement consists of a linear term (with constant strain $\varepsilon(\vec{u})$) plus a Y-periodic "microscopic fluctuation" term, \vec{w} (with zero average strain by periodicity). This assumption of periodicity is reasonable because, by the translational symmetry of an infinite tiling, every cell deforms identically. Now we simply solve for fluctuation \vec{w} putting the microstructure in equilibrium:

$$-\nabla \cdot (C(\vec{y}) : [\varepsilon(\vec{w}(\vec{y})) + \varepsilon(\vec{u})]) = 0 \text{ in } Y,$$

where \vec{y} is the microscopic variable (the coordinate in Y). Then the average stress is $C^{H}: \varepsilon(\vec{u}) = \frac{1}{|Y|} \int_{Y} C: [\varepsilon(\vec{w}) + \varepsilon(\vec{u})] \, \mathrm{d}\vec{y}.$

We can extract the components of C^H by applying it to the 6 canonical symmetric rank 2 basis tensors, $e^{kl} := \frac{1}{2} (\vec{e}_k \otimes \vec{e}_l + \vec{e}_l \otimes \vec{e}_k)$. Each application amounts to solving the *cell problem*:

$$-\nabla \cdot (C^{\text{base}} : [\varepsilon(\mathbf{w}^{kl}) + e^{kl}])) = 0 \text{ in } \omega, \qquad (3.5a)$$

$$[C^{\text{base}}:\varepsilon(\mathbf{w}^{kl})]\hat{\vec{n}} = -[C^{\text{base}}:e^{kl}]\hat{\vec{n}} \text{ on } \partial\omega \backslash \partial Y, \qquad (3.5b)$$

$$\mathbf{w}^{kl}(\vec{y})$$
 Y-periodic, (3.5c)

$$\int_{\omega} \mathbf{w}^{kl}(\vec{y}) \,\mathrm{d}\vec{y} = \mathbf{0},\tag{3.5d}$$

where we rephrased the microscopic force balance as a PDE over ω , since for a structure printed with base material properties C^{base} ,

$$C(\vec{y}) = \begin{cases} C^{\text{base}} & \text{if } \vec{y} \in \omega, \\ 0 & \text{otherwise.} \end{cases}$$
(3.6)

The last constraint in (3.5) eliminates the rigid translation degrees of freedom that still remain after enforcing Y-periodicity.

The homogenized elasticity tensor components are finally just the average over Y of the stress components corresponding to e^{kl} :

$$C_{ijkl}^{H} = \frac{1}{|Y|} \int_{\omega} C_{ijpq}^{\text{base}} [\varepsilon(\vec{w}^{kl}) + e^{kl}]_{pq} \,\mathrm{d}\vec{y}.$$
(3.7)

It is worth noting that C^H does not depend at all on the macroscopic details (shape Ω , force term \vec{f} , or boundary conditions).

FEM implementation. The cell problems (3.5) are solved numerically by a quadratic tetrahedral FEM discretization of ω . The piecewise linear integrand in (3.7) is integrated with exact quadrature.

Given the wire network of the microstructure, defining its topology (Section 3.4), a volume mesh is generated following the STRUT algorithm given in [57]: (i) A polygon is created around both ends of each segment. (ii) For each vertex, the convex hull of the nearby polygons and the vertex is constructed and the polygons themselves are removed from the hull. (iii) For each edge, the convex hull of its two polygons is constructed, and again the two polygons are removed from the hull. A tetrahedral volume mesh is finally created from the resulting closed surface.

The linear elasticity solver must support periodic boundary conditions, which requires the tetrahedral mesh to have an identical tessellation on the opposite periodic cell faces.

Convergence rate. Remarkably, we have observed that the homogenized coefficients remain accurate when the microstructure varies across cells (with few or no repetitions). In our experiments, the deformation behavior of even very coarse tilings closely matches the homogenized behavior (3.14), and we expect similar agreement in general for smoothly varying loads.



Figure 3.14: Deformation of an object with varying material properties per voxel, and the same object with the material in each voxel replaced with the corresponding pattern. The deformed objects are colored by max stress.

3.6 Optimizing Pattern Parameters

An essential step for creating a map of elastic textures is optimizing a pattern with fixed topology to match particular elasticity parameters. This is achieved using shape optimization with respect to the pattern parameters.

The optimization problem. Our goal is to minimize a functional, $J(\omega)$, measuring the difference between the homogenized elastic properties of the pattern and a target elasticity tensor C^* . We choose an objective that is suitable for designing material distributions with large deformations under moderate forces. The distance of compliance tensors, $S^H - S^*$, as opposed to elasticity tensors, is the better choice, since the strain for a constant stress is directly proportional to S^H , not C^H . In fact, minimizing the Frobenius norm of $S^H - S^*$ can be interpreted as a multi-objective least squares optimization to fit the displacements of two cubes—one filled with C^H and the other with C^* —under a set of axis-aligned stretching and shearing loads.

We choose this Frobenius norm as our functional:

$$J(\omega) = \frac{1}{2} \|S^{H}(\omega) - S^{*}\|_{F}^{2}, \qquad (3.8)$$

which we optimize by varying the microstructure shape, ω .

In our case, the microstructure boundary $\partial \omega$ is determined by a small number of parameters \vec{p} , consisting of wire mesh node offsets and thicknesses. While the number of parameters is small, we still expect multiple solutions for minimizing $J(\omega)$ with respect to these parameters. A simple regularization term (staying close to the initial point of optimization) picks a unique solution. We note that instead some quantity of importance (e.g., weight) can be optimized as it is typically done (cf., [117]), adding another nonlinear term to the functional.

The derivative of the boundary $\partial \omega$ with respect to a parameter p_{α} is a vector field $\vec{v}_{p_{\alpha}}(\vec{y})$

defined at points \vec{y} of $\partial \omega$, with $\vec{v}_{p_{\alpha}}(\vec{y})$ being the velocity of \vec{y} if parameter p_{α} changes at unit speed.

Using parameters \vec{p} as variables, the minimization problem can be written as

argmin admissible
$$\vec{p}$$
 $J(\vec{p})$ where $J(\vec{p}) = \frac{1}{2} \|S^H(\vec{p}) - S^*\|_F^2$. (3.9)

The admissibility of parameters is determined by geometric intersection constraints and printability constraints.

The derivative of the objective function with respect to p_{α} can be obtained from $\vec{v}_{p_{\alpha}}$ using the chain rule:

$$\frac{\partial J}{\partial p_{\alpha}} = [S^H - S^*] : \frac{\partial S^H}{\partial p_{\alpha}} = [S^H - S^*] : \mathrm{d}S^H[\vec{v}_{p_{\alpha}}], \tag{3.10}$$

where $dS^{H}[\vec{v}_{p_{\alpha}}]$ is the shape derivative of S^{H} applied to $\vec{v}_{p_{\alpha}}$.

Shape derivative of elasticity tensor The derivative of the microstructure's homogenized elasticity tensor in the direction of shape perturbation \vec{v} is defined as the Gâteaux derivative,

$$dC^{H}[\vec{v}] := \lim_{t \to 0} \frac{C^{H}(\omega(t, \vec{v})) - C^{H}(\omega)}{t}, \qquad (3.11)$$

where $\omega(t, \vec{v}) := {\vec{x} + t\vec{v} : \vec{x} \in \omega}$. As shown in the additional material, the homogenized elasticity tensor, 3.7, can be rewritten in an energy-like form,

$$C_{ijkl}^{H} = \frac{1}{|Y|} \int_{\omega} (e^{ij} + \varepsilon(\vec{w}^{ij})) : C^{\text{base}} : (e^{kl} + \varepsilon(\vec{w}^{kl})) \,\mathrm{d}\vec{y}, \tag{3.12}$$

which is shown to have shape derivative:

$$dC^{H}_{ijkl}[\vec{v}] = \frac{1}{|Y|} \int_{\partial \omega} [(e^{ij} + \varepsilon(\vec{w}^{ij})) : C^{\text{base}}$$

$$: (e^{kl} + \varepsilon(\vec{w}^{kl}))](\vec{v} \cdot \hat{\vec{n}}) \, dA(\vec{y}).$$

$$(3.13)$$

Shape derivative of compliance tensor. The compliance tensor is the symmetric rank 4 inverse of elasticity tensor, i.e. $S_{ijkl}C_{klmn} = \frac{1}{2}(\delta_{im}\delta_{jn} + \delta_{in}\delta_{jm})$. Differentiating and solving for dS^{H} :

$$dS^{H}[\vec{v}] = -S^{H} : dC^{H}[\vec{v}] : S^{H}.$$
(3.14)

Combining the results from 3.10, 3.13, and 3.14, one can compute $\frac{\partial J}{\partial p_{\alpha}}$; the shape derivative and an example velocity field $\vec{v}_{p_{\alpha}}$ are shown in Figure 3.15.



Figure 3.15: Left: a shape derivative, visualized as a steepest ascent normal velocity field for objective 3.8. Right: the shape velocity induced by one of the pattern's thickness parameters.

Numerical computation. The integrand in 3.13 is cubic over each boundary element $(\varepsilon(\vec{w}^{ij}) \text{ and } \vec{v} \cdot \hat{\vec{n}} \text{ are both linear})$, and we use quadrature that evaluates the surface integral exactly. To evaluate the gradient for a given shape we need (a) to mesh the shape (we use the TetGen package [116]); (b) solve 6 periodic elasticity problems to obtain \vec{w}^{ij} , as for homogenization. The cost of a single gradient evaluation (roughly 4.75s on a single core of an Intel Xeon E-2690 v2) is dominated by the cost of periodic meshing and the

elasticity solves, which take roughly equal time. We use the Ceres solver [1]'s Levenberg-Marquardt implementation to minimize the objective; the convergence of the solver is quite fast (Figure 3.16). Typical effects of optimization are shown in Figure 3.17.



Figure 3.16: Convergence of a shape optimization on pattern "(E1,E2) (E1,E4) (E2,E4)." Left: optimization starting point. Right: optimized shape.



Figure 3.17: The path in (E, ν) space traversed by the optimization of pattern "(E1,E2) (E1,E4) (E2,E4)" shown in Figure 3.16. The brown points are intermediate anisotropic microstructures.

3.7 Applications

While our primary focus is on the design of our pattern family and the exploration of its coverage, we demonstrate the application of our elastic textures in two settings: painted material properties and specified deformation behavior. All printing was done using a B9Creator printer at 50 micron resolution with Cherry resin.

Overall workflow. The result of the preceding sections is a lookup map that, for a given (E, ν) , produces an isotropic microstructure with nearby parameters. We assume that we are given a coarse volume mesh filled with identical cube cells.

First, we assign a pair (E_i, ν_i) to each cell *i*, either directly or via material optimization as described below. For each cell, we retrieve a corresponding pattern (topology id, thicknesses, and offsets) from the lookup table. The microstructures in adjacent cells are stitched together by averaging the offsets of each pair of shared face nodes so that they coincide. This might raise the lower node of the pair above some node it supports, n_s , violating printability, but printability can be restored by lowering the pair to n_s 's height.

After this step, the resulting connected wire mesh is inflated with retrieved bar thicknesses, using the process described at the end of Section 3.5. This results in a fine mesh that can be printed.

Material painting. The simplest approach to specifying the material properties (E, ν) is to paint them on a voxel grid. We have created an editor enabling us to paint these layer by layer. The results of fabricating several structures of this type are shown in Figure 3.18.

Material optimization. Manually defining material properties to achieve desired behavior may be difficult, and a more systematic approach is to solve for them. For exam-



Figure 3.18: Examples of objects with painted material properties. All are fabricated with 5mm cells.

ple, consider the following problem: for given applied displacement conditions on some part of the object's boundary, we would like to get some target deformation—e.g., if a bar is compressed along the Z axis, it twists in the X-Y plane—by varying material properties. In other words, we want to find a spatially varying tensor $C_p(\vec{x})$, parametrized by a vector of per-cell isotropic parameters p, such that the following system has a solution:

$$-\nabla \cdot (C_p : \varepsilon(\vec{u})) = 0 \quad \text{in } \Omega$$

$$\vec{u} = \vec{u}_{in} \quad \text{on } \Gamma_{in}$$

$$\vec{u} = \vec{u}_{trg} \quad \text{on } \Gamma_{trg}$$

$$\sigma(\vec{u})\hat{\vec{n}} = 0 \quad \text{on } \Gamma_{trg}$$

(3.15)

where \mathbf{u}_{in} are the applied displacements on compressed area Γ_{in} , and \vec{u}_{trg} are the target displacements of the surface Γ_{trg} on which no forces are applied, as indicated by the last equation. In our implementation, these target/applied conditions can be specified on a per-component basis to set up, e.g., the twisting bar example.

In general, such problems are solved using PDE constrained optimization, requiring solving an adjoint problem at each iteration. However, we found that the following local-global iteration, inspired by related "as-rigid-as-possible" (ARAP) optimization techniques in geometry, works remarkably well. For a fixed $C_p(\vec{x})$, we call \vec{u}_D the solution of the "Dirichlet problem," with the condition of zero tractions on Γ_{trg} removed. \vec{u}_N is the solution of the "Neumann problem," in which the traction condition on Γ_{trg} is retained, but the Dirichlet condition $\vec{u} = \vec{u}_{trg}$ is removed. We initialize the elastic tensor $C_p(\vec{x})$ to a constant. The iteration consists of two steps:

- 1. Solve the Dirichlet and Neumann problems with the current elasticity tensor, to obtain \vec{u}_D and \vec{u}_N .
- 2. Update C_p , minimizing the following energy:

$$\min_{p} \int_{\Omega} \|\varepsilon(\mathbf{u}_{D}) - C_{p}^{-1} : \sigma(\mathbf{u}_{N})\|_{F}^{2} \,\mathrm{d}V$$
(3.16)

This energy can be minimized per-cell for a truly local-global method; however, in practice we find it desirable to regularize p with a Laplacian term, which requires that the "local" step be replaced by a global, but still quadratic optimization. The convergence of this method is shown in Figure 3.19.



Figure 3.19: Convergence of material optimization.

We have used a number of simple voxelized shapes and created a variety of deformation behaviors shown in Figure 3.20. Finally, we have also generated a set of anisotropic samples, with controlled anisotropy ratio, one of which is shown in Figure 3.21.

3.8 Conclusions

We have presented a family of tileable and printable patterns that can be used to approximate varying isotropic material properties. The family has proved useful on a number of simple shape optimization examples: remarkably, all examples in Section 3.7 worked as predicted by simulation without requiring much tuning.

Limitations. There are several limitations of our pattern family. First, some parts of the (E, ν) space are poorly covered. While it is difficult to predict which part of space is theoretically reachable, we conjecture that the space may be significantly broadened. All our simulations and constructions work in the linear regime, not taking into account, e.g., the potential for pattern buckling or other damage. Fortunately, isotropy is correlated with sufficiently high shear modulus, which makes the patterns less prone to buckling. Nevertheless, including this and other nonlinear effects in pattern design is important.

For practical use, it is difficult to restrict the tessellations of objects to equal sized cubes (though one can construct cut cells covered with relatively soft skin). A desirable solution would be to allow patterns to distort to fill arbitrary reasonably well-shaped hex cells.



Figure 3.20: Examples of objects with optimized material properties. All are fabricated with 5mm cells.



Figure 3.21: Compression of an anisotropic sample along the X, Y, and Z directions.

Chapter 4

Mesh Arrangement for Solid Geometry

This chapter is based on our publication [144] in collaboration with Eitan Grinspun, Denis Zorin, Alec Jacobson. My contributions include development of mesh arrangement system jointly with Alec; extensive testing of our algorithm on 10,000 testing models; comparison with all mesh boolean and mesh repair softwares available to us.



Figure 4.1: Our method takes as input any number of meshes (three shown in this 2D illustration). We resolve intersections and assign a winding number vector to every delineated cell. Different boolean operations are achieved as extractions according to these winding number vectors.

4.1 Introduction

Geometric modeling tools are more accessible than ever, scanning technologies are available at the commodity level, additive fabrication technologies rapidly grow in popularity, and platforms have emerged for sharing 3D models online. Unfortunately, the resulting wealth of 3D models comes with a catch. The diversity of these models coincides with unpredictable mesh quality and structure. For example, manually sculpted models are created using a broad range of software by designers with varying skill or intention.

Meanwhile, geometry processing operations increase in sophistication: from remeshing to physical simulation. Yet, many, if not most, available algorithms impose strict requirements on their inputs. As the complexity and amount of 3D data grow, manual one-by-one preprocessing is no longer acceptable. For example, cumbersome mesh cleanup of solid mesh may take more time than a subsequent physical simulation of it.

An important class of mesh repair and solid operations aims to preserve input geometry as much as possible when recombine it in new ways or converting to suitable form for a downstream application. These operations are elegantly viewed as operations on space partitions defined by *mesh arrangements*. A *mesh arrangement* is a collection of (possibly non-manifold, open-boundary, self-intersecting, with degenerate triangles, etc.) meshes, partitioning the space into a number of cells. This view seamlessly unifies tasks often viewed as distinct, such as mesh repair and boolean operations.

We introduce mesh arrangements constructed from a restricted class of meshes: those with *piecewise-constant winding number*, or PWN. By surveying 10,000 popular meshes, we demonstrate that PWN meshes cover a large fraction of practically relevant situations. We define arrangements of PWN meshes, a lightweight yet powerful representation. Casually, a PWN mesh arrangement is composed from possibly (self-)overlapping components bounding a number of solids. These arrangements enable a set of highly *robust* and *conservative* algorithms. By robust, we mean that algorithms successfully produce output for *all* PWN meshes. By conservative, we mean that arrangements preserve original mesh geometry exactly.

Our general approach can be separated into two stages: adding meshes to an arrangement independent of the desired operation—and extracting the boundary of the result according to an extraction function describing the desired boundary in terms of the winding numbers of the region it bounds (see Figure 4.1).

The first stage, in turn, consists of resolving intersections, partitioning space into cells, and labeling each cell with winding numbers with respect to each input mesh.

In the second stage, all classic boolean operations (e.g., union, intersection, difference) have trivial extraction function definitions. Beyond those, we explore other interesting functions, such as extracting the "self-union" of a single, overlapping input mesh or extracting all regions of space inside at least two of many input meshes.

Our method guarantees as output a *solid* mesh. Informally, a solid mesh is the nondegenerate boundary of a solid subregion of \mathbb{R}^3 . Our method also guarantees that the output is *exact*, i.e., interpreting the input positions as exact rationals, all intersections result from exact construction. The output coordinates may optionally be converted to floating point in a post-process.

We validate our method on the 10,000 triangle meshes from the online 3D printing repository *Thingiverse*, as well as benchmarks of previous work. We present extensive comparisons with state-of-the art methods, all of which fail with significant frequency, either rejecting the input, failing to produce any output, or producing an output that is not a solid mesh. We also evaluate the conversion of our exact results to floating-point positions; in this case, we outperform existing floating point methods along the same criteria.

4.2 Related work

Most previous work can be separated into two broad groups: boolean operations on different classes of objects and "mesh repair", in particular, elimination of self-intersections, computing outer hulls and similar operations. While both require intersecting meshes or surfaces, in most cases the problems have been treated disjointly.



Figure 4.2: Self-intersections are not just "artifacts." They also occur intentionally during modeling. A coffee mug handle is extruded then curled inward on itself (blue). The self-intersections (orange) do not prohibit constructing the shape's underlying torus-topology self-union with our algorithm (green).

Previous boolean methods define a restrictive class of input 3D pointsets that are closed with respect to set operations. The methods output a restrictive class of boundary representation or spatial partition. In almost all cases, it is assumed that creating a valid boundary representation from a *broader* class of inputs is a separate task, delegated to the user or preprocessing. Namely, inputs not meeting the strict requirements are not handled directly.

Boolean operations Previous works differ by generality of input/output representations and tradeoffs between performance and robustness. We compare our representation and algorithms to the state of the art in Sections **??** and **??**. For now, we categorize approaches, highlighting salient similarities and distinctions.

The current standard in robustness is CGAL's exact-arithmetic implementation [54] of Nef polyhedra [23]. CGAL's implementation requires a valid Nef polyhedron data structure (Sphere Maps and Selective Nef Complexes) as input, with currently available tools in CGAL and OPENSCAD (a modeling tool bootstrapping CGAL) only allowing conversion of embedded polyhedra to this form, excluding inputs with self-intersections, non-manifold features, and inner cavities, although the latter two could be represented by the Nef representation. More fundamentally, the generality of the representation requires a complex heavy-weight data structure and has a significant impact on performance.

On the other extreme, Douze et al. [45] assume inputs in the restricted form of embedded

polyhedra with vertices in general position, but is extremely efficient and is capable of handling very large meshes. Douze et al. introduce the concept of variadic boolean operations: immediately evaluating boolean operations involving many inputs rather than decomposing into a tree of binary operations.

Recently, Barki et al. [16] use a more general yet lightweight representation and exact rational arithmetic to handle a variety of near degeneracies. Their efficient algorithm maintains robustness on a 26-model benchmark.

All methods so far are similar to ours in that they add intersections of boundary representations, and proceed to classify elements of the boundary to construct the result. However, these algorithms assume that input surfaces are free of self-intersections. Self-intersections are not only a ubiquitous meshing artifact, but also a common way to model interesting topologies Figure 4.2.

Many early methods based on intersection and classification suffered from robustness problems, before exact-arithmetic based methods became practical. This leads to development of more robust approaches, initially based on conversion to volumetric representations, starting from [91]. The downside of these techniques is that the approximation of the original meshes depends on the chosen resolution level for the volume grid, and high accuracy requires high tessellation. Different approaches were used to accelerate this approach, reduce complexity of the output (e.g. using adaptivity [133]), and make it possible to preserve the original mesh as much as possible [101, 136, 142]. The fundamental issue with this flavor of technique is the approximate and grid-dependent nature of volumetric calculations: while increasing robustness, these may lead to hard-to-control topology changes and deviations from the original geometry.

The space-partition view of boolean operations has appeared most clearly in binary space partitioning (BSP) methods, starting with [124, 97]. Bernstein & Fussell [20] combined this with robust predicates to develop an efficient and robust way to compute booleans on surfaces in BSP representation. As in most other works on booleans, conversion to



Figure 4.3: Combinatorially, the multi-component shapes on the left and right are the same, though their outer hulls (thick) differ.

this representation is viewed as a preprocess, with the range of inputs this preprocess can handle not precisely defined. While compared to volumetric-grid approaches, BSP methods increase mesh complexity more moderately and input geometry is better preserved, yet significant refinement is still needed. Campen & Kobbelt [31] localized their BSPbased method using an octree and perform refinement only locally near intersections. Importantly, this work points out that BSP trees provide a general representation of space partition that flexibly performs both boolean operations, outer hull computations, and other operations. We expand this idea, but use a higher-level and more compact representation of space partition.

Mesh repair Mesh repair techniques historically deviate from boolean operations by focusing on converting a maximally broad range of input meshes to a normalized representation (e.g., closed manifold meshes without self-intersections). These methods often rely on volumetric approximation and for certain problems (e.g., hole-filling) this may be unavoidable. If possible, preserving original geometry is desirable. A common example of mesh repair of this type is computation of the outer hull, though both state-of-the-art methods [31, 9] do not appear to disambiguate nested and non-nested components (see Figure 4.3). For some problems, such as preparing models for 3D printing, the outer hull



Figure 4.4: The outer-hull (e.g., [9, 31] is simpler to extract, but not always appropriate. Hollow cavities should be maintained for 3D printing not only to reduce material costs, but to maintain functional properties like balancing.

may be inappropriate as it removes inner cavities (see Figure 4.4).

Approaches to robustness, and sources of non-robustness Many codes (e.g., [19, 82, 45]) explicitly assume general position of inputs (no four points on a circle, no co-planar intersections, etc.) and do not attempt to handle numerical non-robustness. The development of new boolean and mesh repair techniques was driven, to a large extent, by robustness considerations. We consider more explicitly how robustness was addressed in different contexts, and the unresolved problems we are addressing in our approach.

The most comprehensive approach is to represent all objects using exact arithmetic. With advances in filtered predicates for efficiency, this approach is increasingly preferred. We (like others [54, 16]) largely follow it. Earlier BSP-based work used the observation [122] that representing points as the intersection of original planes eliminates the need for exact computations (only exact predicates). This, in principle makes it possible to do most computations robustly in floating point, but some constructions or rounding still inevitably appear in all methods, and lead to non-robustness, often in subtle ways.

For example, Banerjee & Rossignac [13] and later Xu & Keyser [139] build topologies using exact computation but construct vertex positions using fixed-precision floating point numbers, leading to self-intersections, inversions (see Figure 4.5) and degeneracies



Figure 4.5: By outputting an exact solid mesh, our method ensures there are no inversions (illustrated as correct signed distance on left). Previous methods may create inversions, and even small inversions catastrophically effect signed distance to the output (right).

Method	CAVITIE	Non- manifoi	Multi- comp.	Exact	Seams	Degen- eracies	Self- inter.	w > 1
CGAL	Х	Х	•	•	Х	Х	Х	Х
[Campen 2010a]	Х	Х	Х	Х	Х	Х	•	Х
QuickCSG	•	Х	Х	Х	Х	Х	Х	Х
CARVE	٠	•	•	Х	Х	Х	Х	Х
Cork	•	•	•	Х	Х	Х	Х	Х
[Bernstein 2009]	٠	•	Х	•	٠	Х	Х	Х
[Barki 2015]	•	•	•	•	•	•	Х	Х
Our method	•	•	•	•	•	•	•	٠

Table 4.1: Mesh boolean algorithm input preconditions feature chart: Previous techniques fall short have severe input restrictions.

in the output. Campen & Kobbelt [31] round all input vertices aggressively to ensure exact plane intersections for a BSP-representation. We discuss several other problems in existing techniques in greater detail as we describe our method. For our approach (and all exact methods), conversion of the output to a floating point (if such a conversion is desired) is a potential source of problems, although we have observed it in a very small proportion of cases and attack it with an additional heuristic utilizing our core method in Section 4.6.1.

4.3 Concepts

The key element of our method is construction of a data structure representing the mesh arrangement, consisting of cells annotated with winding numbers, patches and their adjacency graph, that allows us to extract results of a variety of operations from the arrangement.

This structure is a relatively lightweight representation of a space partition (compared, e.g., to a BSP tree) as it is based on compound surface objects (patches) and allows for complex cells (does not require them to be convex or even topological balls). Yet, it allows us to perform all operations robustly and efficiently.

The inputs to our arrangement creation algorithm are n piecewise-constant winding number triangle meshes $\mathcal{A}_1, \ldots, \mathcal{A}_n$. For extraction of the results of specific operations from the arrangement we use an variadic extraction function f used to determine the solid mesh boundary of which region(s) of space carved out by $\mathcal{A}_1, \ldots, \mathcal{A}_n$ to output.

4.3.1 Piecewise-constant winding number meshes

A triangle mesh is a set of 3D vertices (some of which may be geometrically coinciding) and a set of triangles connecting these vertices, each triangle represented by a triplets of vertices, with orientation implied by the vertex order for non-degenerate triangles. We may view triangles as triplets of vertices as well as pointsets in 3D. When there is an ambiguity, we call the former *combinatorial triangles* and the latter *geometric triangles*. The same applies to edges and vertices: a combinatorial vertex is a vertex index, and a combinatorial edge is a pair of vertex indices.

Effectively, any valid triangular mesh in Wavefront OBJ-like format, is a valid input, subject to one general condition: we require that triangle meshes A_i induce a piecewise-



Figure 4.6: Our assumptions allow a wide class of inputs with self-intersections, multiple components and non-manifold connections, but does not include open boundaries or non-manifold flaps.

constant integer generalized winding number (PWN) field w_i [64]:

$$w_i(\mathbf{p})\mathbb{Z} \qquad \mathbf{p}\,\mathbb{R}\setminus|\mathcal{A}_i|,$$
(4.1)

where $|\mathcal{A}_i|$ denotes the union of all triangles of \mathcal{A}_i viewed as point sets. For a triangle mesh, this is simply the sum of the signed solid angles $_t(\mathbf{p})$ of each oriented triangle t:

$$w_i(\mathbf{p}) = \frac{1}{4} \sum_{t \in \mathcal{A}_i} t(\mathbf{p}).$$
(4.2)

We call meshes with this property *piecewise-constant winding number meshes* or PWN meshes.

A PWN mesh \mathcal{A}_i can be interpreted as dividing all of \mathbb{R}^3 into regions that are outside $(w_i = 0)$ or inside $(w_i 0)$ of the "solid implied by \mathcal{A}_i ." This allows multiplicity $(|w_i| > 0)$ for parts of space considered to be twice, thrice, etc. inside and also allows for negative insideness $(w_i < 0)$ for parts of space inside an inverted part of \mathcal{A}_i (see inset).

PWN meshes may exhibit commonly witnessed "artifacts" making them unsuitable for previous algorithms (see Table 4.1 and Figure 4.6).

NON-MANIFOLD: PWN mesh connectivity may be non-manifold at vertices or edges.

COPLANAR, DUPLICATE: Co-planar and duplicate facets (i.e., geometrically identical, but logically distinct: either belonging to different meshes, or using vertices with the same positions) do not necessarily invalidate a PWN mesh. For example, the conjoining of two cubes abutting along a triangle is a PWN mesh. The conjoining of two entirely identical cubes is a also PWN mesh. However, a cube with a single duplicated triangle is not a PWN mesh.

CAVITIES, MULTI-COMP., |w| > 1: The winding number elegantly handles correctly oriented boundaries of multiple connected components or nested shells, such as a hollowedout sphere with an outer boundary and inversely oriented boundary of the inner cavity. If the inner boundary were not inversely oriented then the core has w = 2 > 1 and is considered *twice* inside.

SEAMS: A mesh with a combinatorially open boundary does not necessarily imply that it is not PWN. Open boundaries are permissible so long as they meet up geometrically along seams.

EXACT: As discussed in Section 4.5, the vertices of PWN mesh may be defined with rational coordinates, not just floating-point.

SELF-INTER.: A PWN mesh may have structured self-intersections. For example, two overlapping spheres constitute a valid PWN mesh. Similarly, a vertex-displacement of a PWN mesh without seams is also a PWN mesh regardless of any incurred selfintersections [107]. We will say that a mesh is *free of self-intersections* if any two geometric triangles of the mesh intersect only over a (combinatorially) shared edge or vertex, or are combinatorially identical. We allow duplicate combinatorial triangles, as these are needed to handle a variety of degenerate configurations correctly.

DEGENERACIES: Geometrically degenerate triangles (zero area) do not affect the winding number. This implies that one-dimensional "needles" will be ignored entirely. One can formalize ignoring degenerate triangles as reconstructing a discretization of the discontinuity sheets of the winding number field. For purposes of boolean set operations, we operate on open-set interiors without boundary.

Verification We may verify whether a mesh \mathcal{A}_i is PWN by resolving self-intersections (see Section 4.5.1) and then checking that the total signed incidence of every edge in the result is zero. For any edge $e = \{i, j\}$ an oriented triangle $f = \{i, j, k\}$ contributes +1 to the total signed incidence of e. An oppositely oriented triangle $g = \{j, i, \ell\}$ contributes -1.

4.3.2 Variadic extraction function

In general, the extraction function f takes as input a winding number vector $\mathbf{w} = [w_1, \ldots, w_n]$ corresponding to the winding number of each input mesh at the points of a given cell of the space partition defined by the mesh arrangement. The function f returns "true" if a region with this winding number vector is to be included in the output, and "false" otherwise.

For example, to implement n-way union, one would provide:

$$f_{\text{union}}\left(\mathbf{w}\right) = \begin{cases} \underset{i \neq w_{i} \neq 0}{\text{if } i \mid w_{i} \neq 0}, \\ \\ \text{false otherwise.} \end{cases}$$
(4.3)

When n = 1, this function will identify a mesh's *self-union*.

Similarly for n-way intersection:

$$f_{\text{intersect}}\left(\mathbf{w}\right) = \begin{cases} \text{true} & \text{if } \widetilde{w_i 0 \ i}, \\ \text{false} & \text{otherwise.} \end{cases}$$
(4.4)

Some extractions are asymmetric, e.g., subtraction $(\mathcal{A}_1 \setminus \mathcal{A}_2)$:

$$f_{\text{minus}}(\mathbf{w}) = \underbrace{w_1 0}_{\text{inside of } \mathcal{A}_1} \text{ and } \underbrace{w_2 = 0}_{\text{outside of } \mathcal{A}_2}$$
(4.5)

One can also design more esoteric functions, such as extracting all parts of space inside at least two of the inputs:

$$f_{\min-2}(\mathbf{w}) = \begin{cases} \text{true} & \text{if } i \text{ and } ji \mid w_i, w_j 0, \\ \\ \text{false} & \text{otherwise.} \end{cases}$$
(4.6)

Changing the two-sided inequalities above (e.g., $w_i 0$) to single-sided inequalities (e.g., $w_i > 0$) results in orientation-sensitive operations [31, 64]. Orientation sensitivity is useful in some cases, where inversion more intuitively denotes the exterior or void space of an input shape.

4.3.3 Solid meshes

Our algorithm's output meshes belong to a special subclass of PWN meshes that we call *solid meshes*. Solid meshes are free of self-intersections, degenerate triangles or duplicate triangles, and their generalized winding number field is either zero or one.

Note that even if the input meshes \mathcal{A}_1 and \mathcal{A}_2 are manifold polyhedra, the output of $\mathcal{C} = \mathcal{A}_1 \mathcal{A}_2$ may be a non-manifold solid mesh (e.g., if \mathcal{A}_1 bounds the unit cube and \mathcal{A}_2 bounds the unit cube offset by (1, 1, 0)then \mathcal{C} will contain a non-manifold edge where \mathcal{A} and \mathcal{A}_2 "kiss", see inset).



4.4 Overview

Before worrying about details of the method, we review the key aspects

of each stage. For now, we consider the usual *binary* boolean operations on two meshes \mathcal{A} and \mathcal{B} . The insets in this section illustrate the stages of the computation of the asymmetric difference $\mathcal{A} \setminus \mathcal{B}$.

Arrangement construction In the first stage, we resolve all intersections between input meshes using exact arithmetic. We add new triangles by subdividing the inputs so that all intersections occur exactly at edges and vertices. All refined triangles retain references to the original triangles of A and B.

In the second stage, we determine adjacency information between *cells* defining a space partition. We organize the mesh resulting from resolving intersections in the first stage into *patches* of triangles connected by manifold edges. By definition, patches are incident to each other along non-manifold mesh edges. Two cells are adjacent via a shared oriented boundary patch. Two patches incident on the same non-manifold edge *may* bound the same cell. We determine the patchcell relations by sorting facets from all incident patches *around* this edge. In this way, we determine the cell adjacency for each connected component of the adjacency graph of patches. To ensure correct cell adjacency of nested components, we identify a boundary facet of the ambient cell surrounding each component and determine if it is con-

tained in an interior (non-ambient) cell of another component, via point location (see Section 4.5.5). After merging the cell adjacency graphs across connected components, there remains a single ambient cell outside of all components.

In the third stage, we assign winding numbers with respect to \mathcal{A} and \mathcal{B} to each cell, $\mathbf{w} = [w_{\mathcal{A}}, w_{\mathcal{B}}]$.



Figure 4.7: Alternative extractions $(\mathcal{AB}, \mathcal{AB}, \mathcal{B} \setminus \mathcal{A})$ share the same input mesh arrangement.

Having constructed the cell-patch adjacency information in the previous stage, this step is purely combinatorial. The ambient "0"-cell is defined to [0, 0].

Remaining cells are labeled via a traversal of the cells: we add +1or -1 to the winding number of the originating mesh of the patch crossed between cells depending on the patch orientation. For example, consider an unlabeled cell adjacent to a cell labeled [a, b] via a patch originating from input \mathcal{A} . That unlabeled cell will receive [a + 1, b] if crossing *into* the oriented patch or [a - 1, b] if crossing *out*.



Extracting the result We identify desired output cells purely by

their assigned winding numbers. For the difference $\mathcal{A} \setminus \mathcal{B}$, we collect the cells labeled [1,0]. We return the facets of the patches separating these desired cells from undesired cells, reversing orientations if necessary. Different extractions reuse the same intersection resolution, cell adjacency, and winding-number labeling of the first three stages (see Figure 4.7).

4.5 Algorithms

In this section, we consider in detail the algorithms for each of the steps overviewed in Section Section 4.4. We will break each stage into core subroutines. For each subroutine, we will provide preconditions on its input and postconditions on its output. We start with specifying preconditions and postconditions of our method as a whole.

Preconditions The method accepts as input a sequence of PWN meshes, and an extraction function whose arguments correspond to the mesh sequence. The mesh vertex coordinates are assumed to be rational coordinates, a property we call (EXACT). We review exactness in the Appendix. In accepting as input the broader class of exact coordinates rather than floating point values, we accommodate upstream operations, whose output is exact.

Postconditions The output of our algorithm is guaranteed to be an exact solid mesh. As such it is a valid *input* to a downstream application of our own algorithm or another module in CGAL. Observe that while our input preconditions permit self-intersections, and co-planar/degenerate/duplicate facets—by design, these will never occur in our output.

4.5.1 Intersection resolution

The first stage of arrangement construction resolves all triangle-triangle intersections, enriching the mesh combinatorics so that all intersections are exactly represented by shared vertices and edges. We consider all input meshes as a single mesh, i.e., we make no distinction between intersections and self-intersections.

Preconditions The input is an exact PWN mesh \mathcal{A} .

Postconditions The output is an exact PWN mesh free of self-intersections, coincident vertices, and degenerate triangles, inducing exactly the same winding number field as the input mesh. **Algorithm** Self-intersection resolution consists of four steps: (1.0) discard exactly zero area input triangles as they do not affect the winding number, (1.1) compute the intersection between every pair of triangles, (1.2) conduct a constrained Delaunay triangulation for every co-planar *cluster* of intersections, and (1.3) extract and replicate subtriangles within each triangle from its cluster's triangulation.

For now we set aside conservative culling for performance acceleration. We consider all pairs of triangles a and b in \mathcal{A} . The intersection intersect(a, b) between these triangles can be one of the following four cases: empty, a single point, a line segment, or a convex polygon (see Figure 4.8). This intersection must be computed exactly, therefore intersect(a, b) commutes.



Figure 4.8: Blue triangles intersect the green at a point, segments and a polygon (*left*, 3D). Subdivided constraints reduce to coplanar points and segments (*middle*, 2D). The original green triangle is replaced with replications of the green triangles of a constrained Delaunay triangulation (CDT) of the coplanar cluster (*right*, 2D).

Next we replace each input triangle with a triangulation containing those elements resulting from the intersections.

Previous methods construct this triangulation *independently* for each triangle [64, 9, 16], but this approach may introduce inconsistencies between overlapping triangles due to non-general position configurations (see Figure 4.9). Inconsistent triangulations of coplanar intersections result in violating the precondition of the following stages that the mesh is free of intersections as defined in Section 4.3.1.

Instead, we gather *clusters* of triangles connected via non-trivial co-planar intersections (i.e., intersections resulting in convex polygons). By construction all triangles in a cluster



Figure 4.9: Two overlapping, co-planar right triangles (left) admit multiple constrained Delaunay triangulations (CDTs). Independent triangulation could lead to inconsistent CDTs (middle and right).



Figure 4.10: We retain the relationship between the output triangles and the inputs. Because of our exactness, attributes like texture coordinates are losslessly maintained.

share the same supporting plane. We compute a 2D constrained Delaunay triangulation (CDT) of the convex hull of each cluster. The original constraints collected from triangles in the cluster are the points, segments and polygons resulting from intersections with all other triangles in the input mesh \mathcal{A} , as well as the vertex points and edges of the cluster triangles themselves. We further subdivide segment constraints so that all intersections are resolved as constraint Steiner points. Finally, we compute the CDT of the convex hull of these points and segments; no additional Steiner vertices are required.

With CDTs constructed for each cluster, we iterate over each original triangle t to collect its respective subdivisions. We select among the CDT cluster those sub-triangles $\{t_1, t_2, \ldots\}$ whose three vertices, according to exact 2D predicates, are not strictly outside t. We clone each sub-triangle t_i and orient it to match t, again using exact 2D predicates. In order to label winding number vectors (Section 4.5.3), the cloned oriented subtriangle stores a reference (ID) to t. This reference is also useful for applications requiring interpolation of texture coordinates, colors, or other attributes onto the boolean output mesh (see Figure 4.10).

We clean up by purging geometrically duplicate vertices. As we are using exact vertex representation, this can be done efficiently using lexicographical sorting and unique entry extraction from the list of vertices. The result is a possibly non-manifold mesh with possible duplicate triangles, but no self-intersections.

Duplicate triangles need to be retained at this stage, as their removal requires knowledge of the extraction function.

The output mesh has exactly the same winding number field as the input. This immediately follows from the fact that in the result all non-zero area triangles of the original meshes are retained, possibly in the subdivided form as a result of intersection resolution. One aspect of ensuring this is cloning sub-triangles at co-planar intersections. In the inset figure, the orange and blue shapes share a side. If only one set of faces is kept in the output the result is not a PWN mesh.



4.5.2 Partitioning space into cells

The second stage explicitly constructs a set of *cells*. We define a cell as a union of oriented *patches* forming a closed manifold mesh with no self-intersections and its interior. The set of cells forms a space partition.

Each patch is a subset of triangles of the input mesh and inherits their orientation; the patch is a maximal connected set of faces with all edges shared by two faces from the set being manifold. The condition implies that a boundary edge of a patch (if it exists) is a non-manifold junction with neighboring patches.

Boundary patches of a cell may be geometrically coplanar, producing zero-volume cells; by the absence of self-intersections, such patches necessarily consist of single triangles sharing the same vertices.
Preconditions The input is a PWN mesh free of self-intersections, co-incident vertices, and degenerate triangles.

Postconditions The output is a bipartite directed graph encoding $_{cells}$ of cell-patch incidences. Each patch node has one incoming and one \sim outgoing edge to cell nodes, representing the volumetric regions the \sim positive and negative sides of the (oriented) patch, respectively, which \sim we call *above* and *below* cells. In the following, we refer to "patch"



(the combinatorial and geometric data structure) and "patch node" (the bipartite graph node) interchangeably, and likewise for cells.

The output also includes mutual references between patches and input mesh triangles, i.e., each patch node contains a list of triangles, and each triangle has a pointer to the patch it is contained in.

Geometrically, the cells cover all \mathbb{R}^3 . Some cells will have zero geometric volume. These cells are always bound by exactly two clones of the same geometric triangle: i.e. two patches, each with one triangle. There may be many such degenerate cells stacked on the same multiply cloned triangle.



When the input to this stage is the resolved intersection (see Section 4.5.1) of n piecewiseconstant winding number inducing meshes, then the output is denoted a *valid cell-patch data structure*.

Algorithm Our cell partitioning algorithm first separates the input mesh into connected components of triangles. Two triangles are considered connected if and only if they share an edge.

For each such connected component, we construct a cell-patch graph independently of the other components. First, we cluster triangles into patches. Starting with any unassigned

triangle we grow a new patch traversing across manifold edges until the boundary of the patch is either empty or consists only of non-manifold edges.

During clustering, we record, for each non-manifold edge, its incident patches.

The adjacency between patches is encoded as a matrix \mathbf{A} , setting $\mathbf{A}(p,q) = e$. which means that patch p is incident to patch q sharing with it a representative non-manifold edge e. Incident patches p and q may share multiple non-manifold edges, and the choice of representative $\mathbf{A}(p,q)$ is arbitrary.

We now construct the bipartite graph of cells and patches encoding the volumetric partition: while the patches have already been established above, it remains to construct the cells and to add, for each patch p, one outgoing edge to $C^{\uparrow}(p)$ and one incoming edge from $C^{\downarrow}(p)$, the cells above and below p, respectively.

We will traverse *all* patch-patch *incidences* in arbitrary order.

When visiting an incident pair (p,q), we retrieve the representative edge $e = \mathbf{A}(p,q)$. As detailed in Section 4.5.5, we sort the *e*-incident $c_{c^+}^+ r_2 \longrightarrow r_1 c_{c^+}^+ c_{c^+}^+ r_2 \longrightarrow r_1 c_{c^+}^+ c_{c^+$

Suppose the sort results in the ordering $[p, r_1, r_2, \ldots, r_k]$, so that r_1 and r_k are immediately "above" and "below" p, respectively (see inset).

If we think of this sorting order as "upward," then each patch's own orientation is either consistent or inconsistent with the sorting order (e.g. in inset, r_2 is inconsistent). Let $C^+(r_i) \equiv C^{\uparrow}(r_i)$ and $C^-(r_i) \equiv C^{\downarrow}(r_i)$ if patch r_i is oriented consistently with the sort, otherwise let $C^+(r_i) \equiv C^{\downarrow}(r_i)$ and $C^-(r_i) \equiv C^{\uparrow}(r_i)$.

We now propagate cell assignments by iterating over each consecutive pair of *e*-incident patches in order, beginning with (p, r_1) and ending with (r_k, p) . When we visit the pair (r_i, r_{i+1}) , our task is to identify the cell references $C^+(r_i) \equiv C^-(r_{i+1})$. If both $C^+(r_i)$ and $C^-(r_{i+1})$ are unassigned, we assign them both to a newly created cell node. If only one is unassigned, we set it to the other cell node. If both are assigned to distinct cell nodes, we merge the two cell nodes of the bipartite graph. After visiting all patch-patch incidences, the bipartite graph for one connected component is complete.

After completing each connected component, it remains to merge the k_2 $L(\mathcal{K}_2, \mathbf{p}) \neq A(\mathcal{K}_2)$ bipartite graphs. In particular, the ambient cell of a *nested* component must be equated to the corresponding internal cell of the *enclosing* component (see inset). As a special case, the ambient cells of all nonnested components must be equated.

As detailed in Section 4.5.5, for each component \mathcal{K}_i , we identify its ambient cell $A(\mathcal{K}_i)$.

We now iterate over each component, determine whether it is nested, and find the cell to which its ambient cell should be equal. When we visit component \mathcal{K}_i , we select an arbitrary point $\mathbf{p} \in \mathcal{K}_i$. We build a set of candidate enclosing components $E = \{\mathcal{K}_j \mid j \neq i, \mathbf{p} \notin A(\mathcal{K}_j)\}$ consisting of every component $\mathcal{K}_j \mathcal{K}_i$ whose ambient cell does not contain **p**. To determine whether a cell contains **p**, we use *point location* (Section 4.5.5), which given a point **p** and component \mathcal{K}_j returns the containing cell $L(\mathcal{K}_j, \mathbf{p})$ and distance from **p** to \mathcal{K}_j .

If the set E is not empty, then \mathcal{K}_i is a nested component. Among the candidates E, we select the one (and only) component \mathcal{K}_j closest to \mathbf{p} . We merge the bipartite graph nodes $A(\mathcal{K}_i)$ and $L(\mathcal{K}_j, \mathbf{p})$, equating the ambient cell of the nested component to the interior cell of its enclosing component, respectively.

If the set E is empty, then \mathcal{K}_i is not a nested component. In this case, we equate its ambient cell $A(\mathcal{K}_i)$ with the universal ambient cell C_0 , defined as the cell containing "all points at infinity", by merging these two nodes in the bipartite graph.

After we have processed all components, the ambient cell of each nested component has been equated with the interior cell of its enclosing component, and the ambient cell of all non-nested components is C_0 . The bipartite graph is now connected; each patch is incident to two cells in a consistent manner.

4.5.3 Winding number labeling

In the third stage, we compute the winding number of each cell with respect to each input mesh.

Preconditions The input is a valid cell-patch graph. The universal ambient cell is seeded with a known winding number vector; by default $\mathbf{w} = [0, ..., 0]$, signifying that infinity lies *outside* all shapes. Only the combinatorial (not geometric) aspects of the input are considered by this algorithm: instead of computing winding numbers geometrically, we use property that the winding number changes by 1 or -1, whenever a surface is crossed, and thus can be computed by propagation along the cell-patch bipartite graph.

Postconditions The output is a valid cell-patch data structure with consistently labeled winding number vector for each cell. Neighboring cells will differ in winding number vector by exactly +1 or -1 in a single entry corresponding to the originating mesh of the patch between them, signed according to its orientation.

This process assigns winding numbers to zero-volume cells, formed by duplicate triangles; although no points have this winding number, one can view this as the number the interior points would have if the cell boundaries are separated consistently with face ordering.

Algorithm Given cell C with a known winding number vector \mathbf{w}_c , we assign the winding number vector of neighboring cells via breadth first traversal. For each oriented patch p separating cell C from neighbor cell N, if the winding number vector \mathbf{w}_n is still unknown we set it to \mathbf{w}_c adjusted to account for *crossing* patch p, originating from mesh \mathcal{A}_i :

$$\mathbf{w}_n \mathbf{w}_c + s_p[_{i1} \dots _{in}], \tag{4.7}$$

where s_p is +1 if cell C lies above p and N below and -1 if vice versa, and $_{ij}$ is Kronecker's delta. We then add cell N to the queue of cells to process later. When the queue is empty, all cells have been labeled, and the algorithm has completed.

Complements By default, all input meshes \mathcal{A}_i are assumed to represent bounded solids. Under this convention, the winding number at infinity is zero, $w_j(\infty) = 0$. Winding numbers elegantly handle complements by subtraction from 1. If \mathcal{A}_j is the complement of $\mathcal{A}_i = \mathcal{A}_i^c$, then

$$w_j = 1 - |w_i|$$
 or $w_j = 1 - w_i$, (4.8)

depending on whether the complement operator is orientation-insensitive or -sensitive, respectively (see inset).¹



As a consequence, if \mathcal{A}_j represents the unbounded complement of some bounded solid, then the seeded winding number vector at infinity should be $w_j(\infty) = 1$.

In this way the winding number elegantly captures set identities. In particular, we produce exactly the same result for $\mathcal{A} \setminus \mathcal{B}$ and $\mathcal{A} \cap \mathcal{B}^c$.

4.5.4 Operation result extraction

Now that the arrangement data structure is constructed, it allows us to perform arbitrary extraction operations. We extract the triangulated boundary of all cells for which an extraction function f is true.

¹In the orientation-insensitive case, the complement of the complement results in taking the absolute value of the winding number w, preserving orientation-insensitive *insideness* defined as w0.

Preconditions The input is a valid cell-patch data structure with consistently labeled winding number vectors and a predicate function $f(\mathbf{w})$ returning true or false for a given winding number vector \mathbf{w} . As in the previous stage, this stage makes use only of combinatorial (not geometric) aspects of the input.

Postconditions The output is a solid mesh.

Algorithm We flag all cells that pass f, collecting all patches separating a flagged cell from an unflagged cell, and then collecting all triangles of those patches, flipping the orientation of triangles from patches with a flagged cell above and unflagged cell below. We then purge possible boundaries arising from zero-volume symbolic cells. Since these always occur as perfect combinatorial duplicates of single triangles, we need only remove all triangles with zero total signed occurrence: sum of +1 if oriented i, j, k and -1 if k, j, i.

4.5.5 Core low-level subroutines

The robustness of our method rests on the correctness of several core low-level subroutines.

Point location

We are asked to locate in which cell a given point lies. This is a special case of the fundamental *point location* problem in computational geometry. We take special care to solve this problem robustly and in the presence of zero-volume cells (e.g., due to resolved co-planar intersections).

Preconditions The input is a query point $\mathbf{q} \in \mathbb{R}^3$ and a valid cell-patch data structure. The query point \mathbf{q} must not lie exactly on the input mesh. **Postcondition** The output is the unique cell containing the query.

Algorithm Searching over all triangles, we find a triangle t containing the point \mathbf{c} on the input mesh closest to the query point \mathbf{q} . Point \mathbf{c} lies either exactly at a vertex v of t, or else along an edge e, or else within the interior of t (but not on its boundary). None of these cases is trivial. The vertex v or edge e could be a non-manifold junction of many cells, and the triangle t could be from a patch lying deep in a "stack" of zero-volume cells due to duplicated faces.

In fact, only at edges can we robustly determine the symbolic and geometric cell arrangement. The cyclic ordering of cells incident on an edge e is consistent, and since \mathbf{q} does not lie on the input mesh it must lie in one of the incident cells. We insert a dummy facet connecting e and \mathbf{q} into the sorted list of facets incident on ϵ . The next facet after the dummy (or previous facet before) must form a patch bounding the cell containing \mathbf{q} .

The ambiguity in the case where **c** lies within the triangle t arises in the presence of duplicates of the triangle t. Since all duplicates share the same three edges, we choose one arbitrarily as the sorting edge e, and insert a dummy as in the edge case above.

If the closest point **c** lies at a vertex v, we identify a good sorting edge e (i.e., on the convex hull of v and its vertex neighbors) and again insert a dummy as above. Identifying the containing cell of a query whose point of closest approach is a vertex will also arise when identifying the ambient cell of a component. In this case, we project edges incident on v onto the plane formed by $\mathbf{q}-\mathbf{c}$ and any orthogonal vector (rather than the xy plane) and then follow the rest of the ambient cell identification algorithm in Section 4.5.5.

Ambient cell identification

In this subroutine, we identify the ambient cell (containing all points at infinity) of a given mesh.



Figure 4.11: Consider a tetrahedron (orange) inside a wedge (blue), connected at a non-manifold vertex. One face normal of the interior tetrahedron (orange) has a larger *x*-component than the normals of exterior triangles (blue) incident on this non-manifold *outer vertex*.

Preconditions The input is a possibly non-manifold but self-intersection-free triangle mesh and a corresponding cell-patch data structure.

Postconditions The output is a facet guaranteed to contain an outer vertex (a vertex on the convex hull of the vertices) and participate in a patch forming part of the boundary of the ambient cell of this mesh containing all points at infinity.

Algorithm We could solve this problem by identifying the cell containing some arbitrary far away point using the point location algorithm of Section 4.5.5. However, we enjoy the performance benefits of avoiding closest point computation by choosing a query point with a vertex as its known closest point.

We locate a vertex v with the maximum x-coordinate magnitude, breaking ties arbitrarily. It follows immediately that $\mathbf{q} = \mathbf{v} + (1, 0, 0)$ lies in the desired ambient cell and that \mathbf{v} is the point of closest approach of \mathbf{q} to the input mesh.

We will identify the ambient cell by finding a facet incident on v that is part of a patch on the ambient cell's boundary. To find an incident outer facet, we first select an edge incident on this vertex that also lies on the convex hull. Then we sort facets cyclically around this edge and select one of the two facets that are part of ambient-cell boundary patches.

Sorting facets around an edge is discussed in Section 4.5.5, so it remains to identify an edge of the convex hull edge on the vertex with maximal x-coordinate. We rely on our exact representation of the input mesh and the ability to determine predicates exactly (e.g., is a point below, on, or above a plane?). We sort incident edges with respect to their projection on the xy-plane. We select the edge whose projected edge-vector $\epsilon = (e_x, e_y)$ is most orthogonal to the x-axis. Our particular exact representation kernel allows construction of quotients (but not square roots), so we identify the edge with maximum slope as a line function of x: that is, according to $|e_y/e_x|$. We may break ties arbitrarily because all edges with maximum slope must lie on the convex hull.

Remark [9] proceeds in a similar way by finding a maximal x-coordinate vertex and then chooses the incident triangle whose normal has the largest magnitude x-component. This criterion cannot be applied if the vertex is non-manifold: an inner "flap" might have a more outward-pointing normal than the true outer facets. For a concrete counterexample, consider extruding the triangle $\{(0,0), (1,1), (0,2)\}$ two units in the z-direction, then move the lower-right corner to (2,1,0) and add a inner tetrahedron connecting that vertex to the top-left corners and any interior vertex floating below (see Figure 4.11).

Cyclical sort triangles about a common edge

The cell partitioning, point location, and ambient cell identification subroutines depend on the ability to sort triangles about a common edge robustly. We sort only at one representative edge between incident patches, rather than at *every* edge of the triangulation (cf. [9, 16]).

Sorting triangles around a common edge is misleadingly innocuous. This subroutine must (and will) ensure consistent ordering of exactly duplicate triangles (e.g., resulting from resolved co-planar input triangles) and geometrically correct ordering of nearly co-planar triangles.



Figure 4.12: Consider five co-planar facets (lines) incident on two oriented edges (dots). Using the edge orientation to *sign* these indices during sorting ensures that orderings from either edge are consistent (left). Otherwise the sort is the same regardless of the edge orientation leading to inconsistent ordering.

Preconditions The input is a set of m non-degenerate triangles t_1, \ldots, t_m incident on a mutual (non-degenerate) edge $\{i, j\}$. Each triangle t_i is endowed with a globally assigned index i (e.g., its index in the non-manifold output mesh after resolving all intersections). It is assumed that if two triangles are coplanar, then either they intersect only along the edge $\{i, j\}$ (their dihedral angle is 180°) or they are geometrically identical (same third vertex position and their dihedral angle is 0°).

Postconditions This subroutine outputs a sorted (clockwise) ordering of the triangles, looking down the edge $\{i, j\}$. Geometrically distinct triangles are sorted cyclically according to their dihedral angle with the first triangle t_1 . Duplicate triangles—without loss of generality all are $\{i, j, k\}$ —are sorted *consistently* in the sense that their relative ordering is maintained when sorting around $\{i, j\}$, $\{j, k\}$, or $\{k, i\}$ and their ordering is reversed when sorting around $\{j, i\}$, $\{k, j\}$, or $\{i, k\}$.

Algorithm Let us say that each triangle t is *positively* incident on $\{i, j\}$ if $t = \{i, j, k\}$ and otherwise *negatively incident* (i.e., if $t = \{k, j, i\}$).

Let \mathbf{p}_k refer to the vertex position of triangle t_k 's third "flap" vertex not lying on the shared edge $\{i, j\}$. Our recursive divide-and-conquer algorithm begins by selecting a starting triangle $t = t_0$ and sorting each other triangle t_k into one of four groups, based



Figure 4.13: We conduct extensive evaluation of the robustness of our method on a dataset of 10,000 popular real-world models.

on whether \mathbf{p}_k lies (1) co-planar with t_0 and on the same side of $\{i, j\}$ as \mathbf{p}_0 , (2) co-planar with t_0 , and on the opposite side of $\{i, j\}$ as \mathbf{p}_0 , (3) below the plane of t_0 , (4) above the plane of t_0 .

We sort within groups (1) and (2) by simulating simplicity à la [47]. Duplicate triangles are sorted according to their uniquely assigned index $_k$. To ensure that this ordering is consistent and not erroneously reversed when viewed from a different edge incident on the same replicated triangles, we sign these indices based on the signed incidence of each triangle with respect to $\{i, j\}$ (see Figure 4.12). This symbolic perturbation will differ depending on the input indices, but once indices are fixed it is always consistent. Only the ordering of zero-volume cells are effected, so different orderings will always produce the same geometric result.

Triangles in groups (3) and (4) are sorted by recursive calls. The complete output is then simply the merger of the four sorted groups.

4.6 Implementation

We implemented the algorithm in C++ utilizing the exact arithmetic kernel of the popular CGAL library. We specifically use its subroutines for: exact testing and construction triangle-triangle intersections; 2D constrained Delaunay tessellation (CDT); point-triangle closest point queries and point-plane predicates. We found CGAL's CDT implementation to be robust on all examples if constraints are subdivided at intersections as a preprocess.

We also use CGAL's built-in bounding-box-based spatial acceleration for collecting a list of candidate triangle-triangle intersections. We further accelerate the exact triangletriangle intersection detection and construction by processing candidates in parallel. Due to the reference counting employed by CGAL's deferred evaluation exact number type (CGAL::Lazy_exact_nt), seemingly read-only simultaneous access of the triangle data is unsafe. Fortunately intersection detection and construction is compute-bound, so even placing mutex locks around *every mesh vertex* leads to parallelism performance gains.

We also use CGAL's axis-aligned bounding-box hierarchy for point to triangle-soup closest point querying. We further accelerate the point location in Section 4.5.5 by culling points entirely outside of the bounding box of a component (the query point must then lie in that component's ambient cell).

We have integrated our open-source implementation into LIBIGL [65].

4.6.1 Converting to floating-point

While input meshes with floating-point vertex position coordinates losslessly convert to our exact representation, the reverse is not true about our output exact meshes. Naively *rounding* a solid exact mesh to floating point may result in a non-solid mesh due to newly introduced self-intersections. This occurs in 2.19% of our output meshes in the Thingiverse dataset.

In Computational Geometry, this problem is known as *vertex rounding*. Without allowing subdivision of facets and insertion of new vertices, this problem is NP-hard [84]. Allowing for re-triangulation, a robust—albeit slow and complicated—solution to this problem exists *in theory* [52].

To fit into floating-point pipelines and fairly compare to previous methods producing

Cleanliness of the	he 10,000	Thingiverse	meshes
--------------------	-----------	-------------	--------

no duplicate faces			91.8%
no open boundaries			88.6%
edge-manifold			85.9%
no degenerate faces			83.9%
vertex-manifold		77.	6%
single component		74.5	%
no self-intersections	54.7%		

Figure 4.14: The Thingi10K dataset contains real-world meshes with real-world problems.

floating-point output meshes (e.g. [19, 9, 45]), we propose a heuristic for rounding our exact output meshes to floating-point. Our heuristic is related to the method proposed in [108].

Preconditions We assume the input to be a solid triangle-mesh with exact coordinates.

Postconditions Though we can make no guarantees of convergence, our exact method equipped with this rounding heuristic successfully finds self-intersection free floating-point meshes for 99.95% of the dataset. Otherwise, we can only claim the output to remain a PWN mesh.

Heuristic Given a solid mesh with exact vertices, we iteratively apply the following steps: (1) round all vertices to double precision floating-point coordinates, (2) find all triangles participating in self-intersections (if none, then return), (3) round all vertices of these triangles to single precision floating-point coordinates, and (4) compute the self-union of the resulting mesh.

4.7 Experiments and results

Constructed or procedurally generated examples may help investigate corner cases, but do not necessarily report how robustly an algorithm will perform in practice. To this end, we gather a dataset of 10,000 meshes from "the wild," and test our method and previous works against it. Considering these meshes as a representative sampling of a general population of meshes encountered in practice, we evaluate the restrictiveness of preconditions and the robustness of claimed postconditions across methods.

4.7.1 Thingi10K dataset

Contents and methodology The Thingi10K dataset contains the first 10,000 meshes of "Featured" models on thingiverse.com, a popular shape repository. These models are heavily biased toward models designed by amateurs or semi-professionals for 3D printing (though there is no official restrictive policy). We therefore interpret these models as a representative sampling of the population of meshes *intended* to model a solid 3D object.

Each "Thing" featured on thingiverse.com may contain several distinct mesh files. We collected the first 2011 Things, totalling 10,000 meshes (see Figure 4.13). All Things are released under free licenses (GPL, LGPL, Creative Commons, BSD, or public domain). The original meshes came in a biased variety of file formats: 9956 .stl, 42 .obj, one .off, and one .ply. The vast majority of meshes have single-precision vertex-coordinates. Since .stl files store triangle streams rather than meshes, we immediately merge *exactly* duplicate corners. The number of faces in each mesh follow a log-normal distribution with geometric mean = 5077.6 and geometric standard deviation = $8.5 - \frac{5078}{10} - \frac{5078}{1000} - \frac{5078}{1000}$ (see inset).

Postconditions, self-union of 8616 meshes



Figure 4.15: Previous methods frequently failed to produce an output without selfintersections, without open boundaries, and with piecewise-constant winding number.

 10,000 meshes

 Comparing preconditions
 Of the 10,000 meshes, 8616 meet 8616 pwn

 our PWN precondition. Of these, 5113 are solid meshes, and of

 those 4963 are manifold polyhedra.

The 10,000 meshes exhibit a variety of typical problematic cases: open boundaries, self-intersections, non-manifold ele-

ments, multiple components, etc. (see Figure 4.14). Among the 4524 meshes containing self-intersections, 3082 contain coplanar self-intersections. This quantifies an approximation of the fraction of models deviating from the general positioning assumption.

polyhedra

Many "problematic" meshes seem to result from modeling with self-intersections (see Figure 4.2) and overlapping, independently modeled components or from previous failed boolean operations.

4.7.2 Testing self-union

Assuming each mesh in the Thingi10K dataset to represent an *intended solid*, we compare extracting a valid boundary of this solid with available implementations of five previous works: "CGAL" [34], "Carve" [33], "Cork" [19], "QuickCSG" [45], "Attene" [9].

We emphasize that the results of experimental comparison in this case reflect both algorithmic limitations and implementation deficiencies, so a different implementation of any given method could potentially perform better. The no-longer-maintained implementation of [20] failed on most examples. Similarly, the web-service implementation of [31] failed to produce a result roughly 40% of the time. We are unable to obtain implementations or outputs for other methods (e.g. [16]).

We limit our comparison to the 8616 PWN meshes. Of these, only 3413 contain selfintersections. Nonetheless, we consider all 8616 PWN meshes as implementations relying on internal rounding (e.g. [9, 19]) often also stumble on *nearly* self-intersecting meshes.

Attene's mesh repair method computes the outer hull, rather than the self-union [9]. The other implementations do not provide an explicit API for conducting self-union, so we intersect the input model with its conservative bounding box.

Comparing postconditions The expected result of the self-union operation is a solid mesh. We report whether the tested methods successfully produced an output and whether that output met certain necessary postconditions. Testing whether the output mesh is solid requires a correct implementation of cyclic facet ordering around a non-manifold edge to determine that all incident cells are alternating zero to one winding number. Absent trusted third-party code, we test for necessary (but not sufficient) conditions: lack of self-intersections, lack of open boundaries, lack of non-zero total signed incidence edges. Such meshes are a strict subclass of PWN meshes, but a superclass of solid meshes. Our exact method succeeds with 100% success rate across all criteria (see Figure 4.15). Previous methods fall short in at least one criteria. This unique success



Running time distributions, self-union of 8616 meshes seconds

Figure 4.16: The performance of our method is competitive with existing floating-point methods and faster than the state-of-the-art exact method [34]. Geometric means given for each method. Unequal histogram areas correspond to success rate.

Performance profile of four-stage algorithm

% of total running time



Figure 4.17: The main bottleneck of our algorithm is triangle-triangle intersection resolution. Within this major stage, the intersection detection and exact construction dominate.

places our exact method robustly into the exact geometry pipeline.

In a floating-point context, our method also out-performs all others. Our heuristic for converting our exact outputs to floating-point meshes in Section 4.6.1 succeeds in removing new self-intersections all but five cases out of the 8616. These meshes fail to converge after 20 iterations. Rates of closedness and total signed edge-incidence are—by construction—maintained at 100%.

The specific causes of failure of the previous methods are difficult to determine. We can identify robustness flaws associated with characteristics of the input meshes. Methods assuming general positioning or resorting to numerical perturbation [19, 45] will struggle in the presence of coplanar intersections.

Attene assumes accurate floating-point normals during self-intersection culling and outer hull extraction [9], but inputs may contain degenerate or nearly degenerate triangles with untrustworthy normals. The inset highlights self-intersections (orange) and an open boundary (red) on a problematic output of Attene's.



Performance We collected timing information across the 8616 self-

unions for our method and four of the other methods (CGAL, Carve, QuickCSG, Cork) locally on a machine with an 8-core Intel Xeon 3GHz processor with 16GB of memory.² The violin histograms of running timings in Figure 4.16 show that while ours is not the fastest, it is competitive.

The Thingi10K dataset also provides means to further examine the performance of our individual subroutines. The profile in Figure 4.17 reveals that resolving intersections is the dominating bottleneck.

²Attene provided results independently.



Figure 4.18: Self-intersections confuse per-vertex ambient occlusion and sharp-line detection. Rendering the outerhull ameliorates this.



Figure 4.19: David emerges from a block by repeatedly subtracting the Minkowski sum of a drill bit along piecewise-linear paths.

4.7.3 General discussion

Outer hull The outer hull of an input triangle mesh is defined as those triangles reachable from infinity by some (possibly non-straight) path that does not intersect the mesh [32, 9]. In general, the outer hull *cannot* be categorized in terms of the winding number: boundaries with inner hollow cavities with zero winding number are not part of the outer hull. For some applications, retaining these inner cavities is crucial. For other applications, such as rendering, the outer hull may be appropriate and desired (see Figure 4.18). We can easily adapt our algorithm to compute outer hulls. We construct cell partition according to Section 4.5.2, find the ambient cell according to Section 4.5.5, and simply extract its boundary as per Section 4.5.4.

Minkowski sums The Minkowski sum of a solid mesh \mathcal{A} along a line segment {s, d} can be computed as the union of \mathcal{A} at s, \mathcal{A} at d, and the union of all prisms formed by triangles of \mathcal{A} along {s, d}:

$$\mathcal{A} + \{\mathbf{s}, \mathbf{d}\} = \bigcup \left(\mathcal{A} + \mathbf{s}, \mathcal{A} + \mathbf{d}, \bigcup_{t\mathcal{A}} t + \{\mathbf{s}, \mathbf{d}\} \right).$$
(4.9)

Explicitly computing the union of all triangular prism via our mesh boolean algorithm would produce the correct result but after too much unnecessary computation: most neighboring prisms are exact duplicates. We cull the union of prisms with a pre-process, removing all facets with zero total signed occurrence, replacing all instances of facets with 2k total signed occurrences with k positive/negative clones. This proof-of-concept inherits the robustness of our method, but is likely suboptimal in terms of performance compared to specialized methods [32]. In Figure 4.19, we simulate a CNC-milling tool.

Traditional binary boolean tests A traditional test for a boolean algorithm is to select two meshes, randomly rotate them, then conduct a binary operation (union, intersection, difference, etc.) and investigate the result for artifacts or errors. This type



Figure 4.20: Numerical perturbation can produce spurious artifacts.



Figure 4.21: We reproduce and exhaustively expand the pairwise testing in [16] (see supplemental material).



Union of each mesh of [Barki et al. 2015] with itself rotated by $\frac{\pi}{10}, \frac{2\pi}{10}, \dots, \pi$

Figure 4.22: We reproduce and expand upon the \mathcal{A} -union-rotated- \mathcal{A} style test in [16].

of testing encourages the general positioning assumption and may give a false sense of robustness in cases with coplanar intersections and exact co-incidences. An extreme case is taking the intersection of an object with a clone of itself (see Figure 4.20). Methods based on numerical perturbation, such as [19], panic in the presence of so many co-planar intersections.

For completeness, we reproduce and expand upon the testing in [16]. Barki et al. compute the union and intersection for 22 pairs of meshes from a collection of 26 standard computer graphics meshes (the Armadillo, the Cow, the Dino, etc.). We exhaustively compute union, intersection, and both asymmetric differences for all pairs (see Figure 4.21). All 4(26(26+1))/2 = 1404 tests result in valid solid meshes. For two of these meshes, Barki et al. also computes the union and intersection of the mesh and a clone rotated by random rotation. For all 26 meshes, we compute the union of the mesh and 10 clones rotated by $/10, 2/10, \ldots$, about the same axis (see Figure 4.22). All our results are valid solid meshes.

The robustness of several previous works rely on the assumption that input vertices lie on a regular grid [31] or at general positions [19, 45]. However, input rounding



Figure 4.23: A popular commercial app produces three different results (presumably due to randomization), yet all are incorrect.



Figure 4.24: Our method supports n-ary operations. For example, extracting all regions inside at least k of the input spheres centered at each corner of the unit cube.

or perturbation—no matter how subtle—may introduce unnecessary intersections that merge disjoint components (see Figure 4.31) or cause numerical problems (see Figure 4.20). In contrast, the exact nature of our approach allows us to only resolve intersections already present in the inputs.

Others (e.g., [16]) have demonstrated robustness issues with boolean implementations in commercial software such as MAYA. We add to this by comparing to Trimble's SKETCHUP PRO. SKETCHUP PRO consistently fails to intersect four randomly rotated icosahedra (see Figure 4.23). We also attempted to union each of the 26 models of [16] with a clone rotated by 18 (a simplified version of Figure 4.22). After a day of computation, only 12 produced an output, and none were without flaws (all were combinatorially open, only two were without self-intersections).

Although very common, inputs with multiple, possibly nested, components are often



Figure 4.25: Simple and complex variadic operations cost the same using our mesh arrangements. Converting variadic operations to a cascade of binary operations is worst-case exponential in time.

overlooked in previous works. The implementation of [31] assumes only single component inputs, and [34] does not detect nested voids automatically. In contrast, our algorithm correctly handles multiple components as illustrated in Figure 4.26.

Stress tests In addition to standard tests on common computer graphics models, we also stress test our algorithm on challenging examples. Our algorithm is robust for carrying out consecutive boolean operations because the output solid mesh is trivially a valid PWN input for the following operations (see Figure 4.27).

An interesting and challenging application of boolean operations is to "undo" boolean subtractions given only the argument and the result, *produced by an unknown boolean implementation* (see Figure 4.28).

Generality Besides conventional boolean operations, the space partition defined by mesh arrangement is useful for many important geometry processing applications. In Figure 4.29, the outer hull computation is a necessary preprocessing step for generating a volumetric discretization for structural analysis [145]. The outer hull is also useful for culling extra internal complexity (see Figure 4.30).

Our variadic formulation also allows us to compute regions inside at least k of the input meshes without the combinatorial explosion associated with binary boolean operations (see Figure 4.24). Figure 4.25 considers ten intersecting tetrahedra. Our variadic union



Figure 4.26: Disconnected stars and overlapping spike components correctly unites with a nested turtle (slice view above).

of all ten tets is roughly twice as fast as decomposing the union into a cascading tree of binary union operations (and $6.5 \times$ faster than a linear chain of binary unions). Intuitively, this is because our intersection resolution is the most *economical* for this arrangement. In contrast, repeated unions will require resolving intersections with previous results, aggregating unnecessary complexity: though geometrically identical, our result has 384 triangles, compared to the cascading tree's 1000. We also construct extraction of the region inside at least five input tetrahedra. Decomposing this into a binary tree of cascading operations leads to an exponential number of operations in the number of tets $((5\binom{10}{5} - 1) = 1259$ binary operations for ten tets). The aggregation of complexity is catastrophic leading to performance measured in hours. Instead, extracting this result from our arrangement requires the same cost as extracting the union: just a few seconds.

Lastly, the cell data structure used by our algorithm can be easily extended for customized applications. For example, it is easy to eliminate small cells immersed inside a shape (see Figure 4.32).

4.8 Limitations & Future work

A limitation of this method is the requirement that the input mesh have no open boundaries or non-manifold "flaps". Of the 10,000 meshes in our Thingiverse dataset, 18% did not meet our preconditions. While repairing invalid input meshes is beyond the scope



Figure 4.27: We reproduce the carving example of [20] by subtracting 10,000 dodecahedra from a box.



Figure 4.28: The groved, yellow frog is the result of subtracting the stripy, blue flog from an unknown (presumably solid) frog using some boolean implementation (not ours). Our robust union recovers the original frog (blue and yellow).

of this method, many previous works are available [8, 30, 64]. On the other hand, the *high-level* structure of our approach would clearly extend to meshes with boundaries and unstructured non-manifoldness via the generalized winding number [64]. However, our *low-level* combinatorial and sorting based subroutines would need to be replaced with robust or exact evaluations of the generalized winding number. While Barki et al. [16] provide a partial solution for simple cases (e.g., clipping a closed model with a plane), a robust solution for arbitrary geometrically open models is elusive.

Our method is variadic, but does not optimize operations based on the request extraction and inputs. For example, consider conducting the 1000-way union of 999 overlapping spheres enclosed and their conservative bounding box. Clearly resolving the intersections between the 999 spheres is overkill. It would be interesting to explore compiler-style optimization of this computation based on the geometry of the input meshes and the given extraction function.



Figure 4.29: Tetrahedralization of the input foot mesh fails due to overlapping input components, but succeeds after self-union. The volume mesh helps analyze the shapes structure.



Figure 4.30: The coin mesh has been modeled with many overlapping components (gold). Constructing the outer hull adds many new vertices, but removes hidden interior geometry. After decimation, the few triangles are well spent on the visible surface (silver). Naive decimation wastes precious triangles on the hidden interior (bronze).

In the hopes of fostering continued work in this direction and more exhaustive testing in geometry processing at large, we release both our code and 10,000-mesh Thingiverse dataset to the community.

Acknowledgments

We thank G. Bernstein for sharing code and M. Attene for testing on the Thingiverse dataset. We thank M. Campen, A. Fleming H. Maia, J. Panetta, R. Sawhney, O. Stein, P. Thamjaroenporn, O. Winn, and E. Yao for early feedback and proofreading. Funded in part by NSF grants CMMI-11-29917, IIS-14-09286, and IIS-17257.



Figure 4.31: The letters are separated overlapping components on the cryptex. Exact union reveals disjoint rings, shown in different colors, but small tolerances between parts cause inexact methods to merge over zealously.



Figure 4.32: The self-union of a wheel of cheese retains its internal bubbles after slicing (intersecting) with a knife (blue wedge). Slicing the outer hull reveals no bubbles. Eliminating small volumes cells in the self-union before extraction, produces a few bubbles.

Chapter 5

Thingi10K: A Dataset of 10,000 3D-Printing Models

This chapter is based on our submission to Symposium of Geometry Processing 2016. It is in collaboration with Alec Jacobson. My contributions include statistical analysis of Thingi10K, ShapeNetCore and MPZ14 dataset; the creation and maintenance of online query interface for Thingi10K dataset.



Figure 5.1: The Thingi10K dataset contains 10,000 models from from featured "things" on thingiverse.com, a popular online repository.

5.1 Introduction and background

The iconic *Stanford bunny*, now 23 years old, has been melted, shattered, and deformed countless times. While mostly a fun subculture, "bunny torture" is also a legacy of an earlier time when few interesting and free 3D models existed. Testing on such standard models persists despite well-known limitations. As Greg Turk, originator of the bunny, advises, "I actually consider the bunny to be *too good* as a test model. It is fairly smooth, it has manifold connectivity, and it isn't too complex" [130].

Oversimplified testing provides a false sense of robustness and causes not only visual artifacts in computer graphics applications, but also fabrication and functionality artifacts when processing geometry intended for 3D printing. Fortunately, 3D models are now abundant. Modern consumer-level 3D printing technologies nurture new communities of professional and amateur 3D modelers, who share and sell 3D-printable models online (e.g., shapeways.com, sketchfab.com, thingiverse.com). This wealth of data also echoes the demand for state-of-the-art processing techniques and automation within 3D printing pipelines.

However, testing remains inadequate. Existing datasets contain only sanitized models (e.g., [3, 71, 93]) or draw from populations containing raw models not specifically intended for printing (rather, e.g., for shape classification [114, 35] or scene understanding [96, 39]).

In this paper, we will show that the characteristics and issues common to 3D printing models are distinct from models intended for visualization. As such, validating geometry processing techniques related to 3D printing requires a new representative dataset. This ideal dataset should encompass the different contextual and geometric characteristics of commonly printed shapes. Characteristics common to 3D printing models should appear with proportional distributions, and characteristics *inconsistent* with models intended for fabrication should be infrequent (e.g., the open boundaries of a video game character's clothing).

We propose a dataset of 10,000 models culled from a popular shape repository for 3D printing enthusiasts, thingiverse.com. Hereon, we refer to our dataset as *Thingi10K*. Beyond collecting tags and class information available online, we analyze geometric characteristics of each model (e.g., manifoldness, lack of self-intersections, genus). We contrast these statistics against existing large datasets and investigate correlations within the data.

Existing datasets. Myles et al. collect 116 models from academic sources (Stanford Scanning Repository [71] and Aim@Shape Repository [3]) to test their parameterization algorithm [93]. These models correspond to *best-case* input due to their extreme cleanliness and general position assumption (i.e., no four points on a circle, no coplanar intersections, etc.). For 3D printing models in the wild, degeneracies, non-manifoldness and self-intersections are abundant, not special cases. Structured modeling and coordinate quantization tends to break rather than fulfill general position assumptions.

Computer vision and machine learning applications demand large scale training datasets. For example, the NYU Depth Dataset collects thousands of depth video sequences of indoor scenes for object classification [96]. The Princeton Shape Benchmark collects 1,814 polygonal models of specific objects (e.g., animals, furniture) from various internet sources for shape classification [114]. More recently, ShapeNet collects more than three million annotated models [35]. The ShapeNetCore subset contains 57,459 single-object models with semi-automatically generated category information. Although models from these datasets resemble physical objects, their geometric characteristics suggest their intention was for visualization rather than fabrication. These datasets are not suitable for testing 3D printing techniques.

In addition to generic datasets, a variety of specialized datasets exist. For example, Lim et al. provide 219 IKEA 3D models for pose-estimation [75]. Recently, Choi et al. released a dataset of 10,000 scanned objects, with a subset of 383 successfully reconstructed 3D models [39]. The Shape Retrieval Contest releases multiple datasets each year to test re-



Figure 5.2: Our online query interface selects subsets of Thingi10K.

trieval algorithms including generic [25, 72], non-rigid humans [103], sketch-based shapes [73, 74], shape correspondences [24], facial expressions [94, 134], and range scans [46]. Our Thingi10K dataset complements these sources by providing a specialized dataset for 3D printing objects.

We are not the first to utilize Thingiverse models for academic purposes. To test a rapid prototyping interface, Mueller et al. consider Thingiverse models, but report that meshing artifacts required manual cleanup before processing [90]. Beyer et al. procedurally collect 2,250 models with specific tags from Thingiverse to test a decomposition algorithm [21]. Buehler et al. manually sift through 25,000 models from search results on Thingiverse to identify 363 models as "assistive technologies" [27]. Beyond testing a specific routine, these works do not analyze low-level geometric characteristics of the collected models. These *collected* datasets are also not publicly available.

Contributions. Unlike previous datasets, our Thingi10K dataset reflects the variety, complexity and (lack of) quality of 3D printing models. It is immediately useful for testing the performance of methods for structural analysis [121, 145, 132], shape optimization [105, 12, 92], or solid geometry operations [144]. Due to its specialized nature and correlated contextual information, we suspect the dataset is also useful for machine learning and data mining algorithms. We compare the collected contextual information and computed geometric properties of our dataset in detail against two existing datasets: MPZ14 and ShapeNetCore. We demonstrate that these represent two extreme cases in terms geometric quality while our dataset provides a mixture of geometric qualities reflecting real-world settings. All data and analysis of our dataset are freely available to the public. To facilitate exploration and future reuse, we provide an easy-to-use online query interface (see Figure 5.2). This interface augments the Thingiverse front-end with our geometric analysis of each model.

5.2 Methodology

Instead of our hiring professional modelers or scanning physical objects, we leverage the availability of 3D models hosted and shared online. Among all 3D shape repositories, we select Thingiverse for its large and active user community, its vast collection of printvalidated designs, and its restriction to open-source licenses.

As one of the largest online shape repositories, Thingiverse hosts more than a million user-uploaded *things*, 3D designs consisting of one or more 3D *models* (i.e., one or more mesh files). As of October 2015, Thingiverse has more than 2 million active users, with 30-40 uploads each week and 1.7 million downloads per month [81]. Thanks to this community, a design is typically not only modeled virtually but also fabricated by one or more users, which provides invaluable real-world validations.

Our Thingi10K dataset consists of 10,000 models (from 2011 things) systematically culled from Thingiverse via web crawling. Rather than randomly sample the entire repository, which may contain bogus models uploaded by inexperienced users or for testing purposes, we focus on things *featured* on Thingiverse. Featured things are entirely and independently selected by Thingiverse staff based on their design, beauty and manufacturability. In a sense, these 10,000 models represent a subset of the top-quality designs on Thingiverse. Thingi10K contains every 3D model of every thing featured by Thingiverse between Sept. 16, 2009 and Nov. 15, 2015.

5.3 Analysis

The 10,000-model dataset comes from 2,011 unique things designed by 1,083 unique users, covering a large variety. Nearly all models are stored as .stl files (9,956); the rest are .obj (42), .ply (1), and .off (1). We analyze both geometric and contextual information of our dataset to illustrate its representational quality and diversity.



Figure 5.3: Percentile plots of vertex and component count.



Figure 5.4: Highest resolution models from each dataset.

5.3.1 Geometry information

We analyze a variety of mesh complexity and quality measures on our dataset of 3D printing models and compare with two existing datasets: MPZ14 (116 models) and ShapeNetCore (2000 models uniformly sampled from 57,459).

Complexity

Complexity of 3D model does not directly correlate with 3D printing cost. We evaluated three different measures to quantify the complexity of our dataset: number of vertices, number of disconnected components and genus.

Figure 5.3 provides the percentile plot of both vertex and component count over each



Figure 5.5: Connected components of Thingi10K models tend to represent salient parts; those in ShapeNetCore are often just disconnected patches (models with most components shown).

dataset. The vertex count plot indicates that the MPZ14 dataset favors moderately high resolution models and excludes extremely low or high resolution models. On the other hand, the distribution over our dataset and ShapeNetCore is similar, with our dataset covering a larger range. Figure 5.4 illustrates the highest resolution model of each dataset.

Many geometry processing algorithms assume input will be processed one component at a time, so it is not a surprise that MPZ14 contains exclusively single-component models. This assumption is not valid in the context of 3D printing, where multiple components could overlap to form a larger shape. Analysing each component separately may lead to incorrect results. Within our dataset 29% of models have more than one component. ShapeNetCore is 83% multi-component, but close inspection finds many models are composed of incoherent patches or isolated faces (see Figure 5.5) In contrast, 3D printing models with high numbers of components in Thingi10K are typically by design, with the base shape naturally decomposing into smaller components.

The genus distribution of our Thingi10K dataset is similar to MPZ14, but our dataset covers a larger range of genus, with the highest genus over 60 times larger than in MPZ14


Figure 5.6: Models with the highest genus from each dataset.

(see Figure 5.6). To avoid confusion, we limit the genus comparison to single-component, closed and manifold meshes. Zero of the ShapeNetCore models meet this criteria.

Mesh quality

Mesh qualities of a dataset play a major role in determining its usability and representation of models *in the wild*. For example, degenerate or sliver triangles will cause poor accuracy in non-robust finite element simulations, and fragile volumetric meshing routines will fail in the presence of self-intersections. It is crucial to understand the mesh quality of real-world input data in order to design robust and practical algorithms. Existing datasets often focus on high-level properties and provide little insight on their mesh qualities. Our analysis aims to fill this gap.

We analyze 13 mesh quality measurements:

Closed: Every edge is adjacent to 2 or more faces.

Oriented: Every non-boundary edge has zero signed incidence. In other words, the number of positively oriented incident faces must equal to the number of negatively oriented incident faces.



Figure 5.7: MPZ14 models are "too clean," whereas ShapeNetCore are unrealistically corrupted in the context of 3D printing.



Figure 5.8: Percentile plots mesh quality measures.

No isolated vertices: All vertices are adjacent to at least one face.

No duplicated faces: There does not exist a pair of faces sharing the same set of vertices.

Vertex-manifold: The one-ring neighborhood of every vertex is a topological disc.

Edge-manifold: Every non-boundary edge must be incident to exactly two faces.

No degeneracy: All faces must have non-collinear vertices. Degeneracy can be checked with exact predicates [113].

No self-intersection: The intersection of any two faces is either empty, a shared vertex, or a shared edge. Exact predicates are necessary to ensure correctness.

No coplanar intersections: No two faces are coplanar and overlapping. This is a strictly weaker condition than "no self-intersection."

Piecewise-constant winding number (PWN): The winding number field at any non-mesh point is piece-wise constant ([144]).

Solid: The input mesh must be a valid boundary of a subspace of \mathbb{R}^3 . Specifically, it must be PWN, self-intersection free and induce a $\{0, 1\}$ winding number field.

Aspect ratio: The aspect ratio of a triangle is the ratio of its circumradius to the diameter of its incircle.

Intrinsically Delaunay: All edges must have non-negative cotangent weights [51].

Figure 5.7 shows the percentage of models that satisfy each of the first 11 quality mea-



Figure 5.9: Models with tag math (left), sculpture (middle) and scan (right).

customizer challenge sketchup halloween supportless education plastic reprap fun customizer skull gears camera sculpture rpg robotics scan vase toolkitchen household puzzle playset space iphone fantasy printbot miniature ultimaker electronics christmas USefUl container castle music animal lamp holder pla monster mount robot dualstrusion lulzbot pla monster math experiment geometry box arduino arduino tinkercad parametric led capturedornament 23d catch light jewelry maker instrument building modular replicatornewmuseumchallenge architecture

Figure 5.10: Thingi10K user tags highlight the dataset's variety.

sures. Figure 5.8 illustrates the maximum, average aspect ratio and the fraction of non-intrinsic Delaunay edges over all models in each dataset. Our analysis shows that MPZ14 has "unrealistically pristine" mesh quality, whereas ShapeNetCore exhibits mesh quality issues not common to 3D printed models, reflecting that it is gathered from a larger space of 3D models.

MPZ14 has *perfect* mesh quality according to seven different measures. In particular, all models are manifold, oriented and degeneracy-free. Because many geometry processing algorithms do not require the input model to be closed or self-intersection free, data from MPZ14 are perfect as proof-of-concept examples. However, their high quality is due to the fact that models were selected not on merits of their shape, functionality or aesthetics, but rather because they meet certain quality criteria or have been sanitized.

On the other hand, ShapeNetCore has very poor mesh quality according to 6 measures in Figure 5.7. Its maximum and average triangle aspect ratios are visibly worse than Thingi10K and MPZ14. This is partially due to the fact that these data are collected directly from the internet, where models were not necessary designed for fabrication purposes. Many existing learning algorithms side-step the quality issues by transforming boundary representations to depth images or bounding box hierarchies [63]. Performing geometry processing algorithms directly on these models is very hard due to poor mesh quality.

In contrast, our dataset offers a curated collection of 3D meshes with a large range of mesh qualities. It contains a significant number of high quality models as well as a non-negligible proportion of models with common mesh quality problems. Due to its large quantity, our dataset is ideal for stress-testing purposes where one can easily select a subset of the data that matches any combination of mesh criteria (Section 5.4). Because all data are sampled from real-world models designed to be 3D printed, our dataset provides an unbiased view of the mesh qualities used in practice. Our analysis could be used to gauge the restrictions posed by various assumptions on mesh quality. For example, an algorithm assuming self-intersection-free input would automatically exclude 45% of inputs, which may not be acceptable in a real-world settings.

5.3.2 Contextual information

Each thing in our dataset is annotated by its original designer. Thingiverse supports three types of annotations: category, subcategory and tags. The first two must be selected from a predefined list of categories, and the last one is a set of free-form texts created by the user. A total of 4892 distinct tags are used in our dataset. Figure 5.10 illustrates the most frequently used tags.

Unlike ShapeNet [35], which focuses on providing categorical annotations specific to object classification purposes, our dataset comes with a rich and diverse set of original tags ranging from the semantics of a 3D model to the printer/material used for fabrication. For example, Figure 5.9 shows all models with tags math, sculpture and scan.

When combined with geometric analysis, our annotations reveal interesting insights un-



Figure 5.11: A soap bubble chair is decomposed and re-oriented by its designer for support-free 3D printing.



Figure 5.12: All 10,000 models come under open source licenses.

available from previous works. For example, a simple frequency analysis indicates Open-SCAD is the most popular modeling tool used by Thingiverse users. Our dataset shows that 98% of OpenSCAD models are closed, while only 91% of SketchUp models and 85% of TinkerCAD models are closed

Furthermore, due its fabrication-focused nature, many uploaded meshes are "printready" in the sense that their orientation and decompositions are designed for optimal printing outcome (See figure 5.11). Recent papers have tried to solve problems such as decomposing a large model to fit in the print volume and finding ideal print orientations [37]. The Thingi10K models, by their intrinsic nature of being successful prints, represent ground truth data.

Lastly, all things are published under one of the open source licenses. Figure 5.12 illustrates all licenses supported by Thingiverse.

5.4 Online query interface

To facilitate our goal of understanding 3D printed shapes, we provide an online query interface, ten-thousand-models.appspot.com for anyone to explore and dissect the dataset. The query terms may consist of one or more clauses. Each clause specifies a single search condition, e.g. "genus;100". Multiple clauses are separated by commas, and the search engine retrieves models that satisfy all search conditions.

Our query interface is very useful in dissecting the dataset based on mesh quality measures. For example, all single-component, manifold solid meshes without self-intersection and degeneracies can be obtained with the query term "num component=1, is manifold, is solid, without self-intersection, without degeneracy". All meshes satisfying these criteria are listed on the result page (Figure 5.2). We also provide an autogenerated python script to batch download results for custom search terms.

Users of our online query interface can view all contextual and geometry model details (Figure 5.13). In particular, we respect the copyright of each model. On the model detail page, we clearly indicate the original author and open source licence of each model. We also provide links to the original Thingiverse pages where the raw data can be obtained.

To demonstrate the power of our online query interface, Figure 5.14 shows some interesting search results and the query used.

5.5 Conclusion

In this work, we present a large-scale annotated 3D dataset based on models used in 3D printing applications. Our dataset consists of 10,000 meshes crawled systematically from Thingiverse. We analyze both the contextual and geometric information of our dataset and compare with two existing 3D model datasets. Our analysis shows our data covers a large range of categories and provides a balanced representation of real-world data in



Figure 5.13: Both contextual and geometric information of each model are available on its model detail page.

terms of mesh complexity and quality. The entire dataset and our analysis are freely available to the public, and we provide a query interface to facilitate the exploration and dissection of our dataset.

Our dataset could be used as input for stress-testing purposes as well as ground truth for learning algorithms. As for future work, we plan to update and increase the size of the dataset over time to reflect the fast-evolving nature of the 3D printing community. Specifically, we would like to include all featured things from Thingiverse and add support for users to suggest additional models for inclusion. We hope our dataset and the accompanying analysis provide an informative summary of 3D printing models and clarify the requirements for geometry processing algorithms to be robust.



Figure 5.14: Our web interface returns subsets of the Thingi10K dataset via text queries.

Chapter 6

Conclusion

In this thesis, I present four different works under the unifying theme of pushing the limit of 3D printing technologies.

In Chapter 2, we tackle the topic of printability. While traditional finite element analysis could be used to model elastic deformation under a fixed load distribution, the usage cases for 3D printed shapes are complex and unpredictable. Our algorithm computes the *worst case* load distribution that causes the most damage to a given shape. The resulting weakness highlights the problematic regions and could be used as guidance for subsequent modification.

Chapter 3 presents a family of tilable and printable patterns called "Elastic Textures" to achieve a large range of isotropic material properties. We also provides an entire pipeline for designing spatially tiling such "Elastic Textures" to achieve a target deformation behavior.

Lastly, many high-level geometry processing tasks depends on the robustness of lowlevel operations. This is especially true for 3D printing applications where the input 3D models deviates significantly in complexity and quality from standard 3D models used in computer graphics research (Chapter 5). In Chapter 4 we describe a robust, general and variadic algorithm for carrying out a family of solid geometry operations. We test our method on 10,000 models "wild" models crawled from Thingiverse, a popular shape repository.

6.1 Unsolved problems and future works

While 3D printing technologies have shown great promises in many industry sectors, we have encountered several shortcomings that may limit or prevent them from being reliable or be applied in large scale. These shortcomings serve as the ground for future research and development directions.

6.1.1 Material properties

Due to the nature of the fabrication process, 3D printed materials are often very anisotropic. Most published works on computational fabrication in computer graphics assume isotropic material properties for simplicity. Extending these works to handle anisotropic material often increases the complexity of the solution and sometimes invalidates key assumptions that render the solution incorrect.

6.1.2 Accuracy

All fabrication methods come with a margin of error. In the case of 3D printing, the margin of error is printer-dependent and changes over time. 3D printer venders often specify the theoretical minimum resolution (e.g. layer thickness, laser radius etc.) of their products, but it has very little to do with the practical resolution and accuracy. Despite that inaccuracy in geometry may significantly change the solution, it is rarely considered in computational fabrication research.

6.1.3 Volumetric meshing

A large number of published work, including ours, relies on finite element simulation to analyze the physical properties 3D printed models. However, how to obtain the volumetric mesh for such simulation is rarely discussed. The state-of-the-art tetrahedral mesh generator, TetGen, falls short when confronted with models used in everyday practice by the 3D printing community. Without a robust method for generating tetrahedron mesh, many algorithms for 3D model analysis and computational design cannot yet match the need from 3D printing industry.

6.1.4 Large scale validation

Although the price for 3D printing has been falling continuously, it is still very hard to conduct large scale physical validation experiments due to the time and efforts involved. Without such large scale tests, it is often difficult to check the robustness and generality of different approaches.

Bibliography

- [1] S. Agarwal, K. Mierle, and Others. Ceres solver. http://ceres-solver.org.
- [2] S. Ahn, M. Montero, D. Odell, S. Roundy, and P. K. Wright. Anisotropic material properties of fused deposition modeling abs. *Rapid Prototyping Journal*, 8(4):248– 257, 2002.
- [3] Aim@Shape. Aim@shape digital shape workbench v5.0 shape repository. http://visionair.ge.imati.cnr.it, 2004.
- [4] G. Allaire. Shape optimization by the homogenization method, volume 146. Springer, 2002.
- [5] E. Andreassen, B. S. Lazarov, and O. Sigmund. Design of manufacturable 3D extremal elastic microstructure. *Mechanics of Materials*, 69(1):1–10, 2014.
- [6] arianaghababaie. Using processing with ember, 2015.
- [7] ASTM. D5023-07 standard test method for plastics: Dynamic mechanical properties: In flexure (three-point bending). In American Society for Testing and Materials, 2007.
- [8] M. Attene. A lightweight approach to repairing digitized polygon meshes. The Visual Computer, 2010.
- [9] M. Attene. Direct repair of self-intersecting meshes. Graphical Models, 2014.

- [10] M. Avellaneda. Optimal bounds and microgeometries for elastic two-phase composites. SIAM Journal on Applied Mathematics, 47(6):1216–1228, 1987.
- [11] M. Bächer, B. Bickel, D. L. James, and H. Pfister. Fabricating articulated characters from skinned meshes. ACM Trans. Graph., 31(4):47:1–47:9, July 2012.
- [12] M. Bächer, E. Whiting, B. Bickel, and O. Sorkine-Hornung. Spin-it: Optimizing moment of inertia for spinnable objects. ACM Trans. Graph., 2014.
- [13] R. P. Banerjee and J. R. Rossignac. Topologically exact evaluation of polyhedra defined in CSG with loose primitives. *Comput. Graph. Forum*, 1996.
- [14] J. Barbič and D. L. James. Real-time subspace integration for st. venant-kirchhoff deformable models. ACM Trans. Graph., 24(3):982–990, July 2005.
- [15] J. Barbič and D. L. James. Subspace self-collision culling. ACM Trans. Graph., 29(4):81:1–81:9, July 2010.
- [16] H. Barki, G. Guennebaud, and S. Foufou. Exact, robust, and efficient regularized booleans on general 3d meshes. *Computers and Mathematics with Applications*, 2015.
- [17] M. P. Bendsøe. Optimal shape design as a material distribution problem. Structural optimization, 1(4):193–202, 1989.
- [18] M. P. Bendsøe and O. Sigmund. Topology optimization: theory, methods and applications. Springer, 2003.
- [19] G. Bernstein. Cork boolean library, 2013. https://github.com/gilbo/cork.
- [20] G. Bernstein and D. Fussell. Fast, exact, linear booleans. In Proc. SGP, 2009.
- [21] D. Beyer, S. Gurevich, S. Mueller, H.-T. Chen, and P. Baudisch. Platener: Lowfidelity fabrication of 3d objects by substituting 3d print with laser-cut plates. In *Proc. SIGCHI*, 2015.

- [22] B. Bickel, M. Bächer, M. A. Otaduy, H. R. Lee, H. Pfister, M. Gross, and W. Matusik. Design and fabrication of materials with desired deformation behavior. ACM Trans. Graph., 29(4):63:1–63:10, July 2010.
- [23] H. Bieri and W. Nef. Elementary set operations with d-dimensional polyhedra. In Proc. IWCGA, 1988.
- [24] A. Bronstein, M. Bronstein, U. Castellani, A. Dubrovina, L. Guibas, R. Horaud,
 R. Kimmel, D. Knossow, E. Von Lavante, D. Mateus, et al. Shrec 2010: robust correspondence benchmark. In *EW3DOR*, 2010.
- [25] A. Bronstein, M. Bronstein, U. Castellani, B. Falcidieno, A. Fusiello, A. Godil,
 L. Guibas, I. Kokkinos, Z. Lian, M. Ovsjanikov, et al. Shrec 2010: robust largescale shape retrieval benchmark. *Proc. 3DOR*, 2010.
- [26] T. Bückmann, N. Stenger, M. Kadic, J. Kaschke, A. Frölich, T. Kennerknecht, C. Eberl, M. Thiel, and M. Wegener. Tailored 3d mechanical metamaterials made by dip-in direct-laser-writing optical lithography. *Advanced Materials*, 24(20):2710– 2714, 2012.
- [27] E. Buehler, S. Branham, A. Ali, J. J. Chang, M. K. Hofmann, A. Hurst, and S. K. Kane. Sharing is caring: Assistive technology designs on thingiverse. In *Proc. SIGCHI*, 2015.
- [28] J. E. Cadman, S. Zhou, Y. Chen, and Q. Li. On design of multi-functional microstructural materials. *Journal of Materials Science*, 48(1):51–66, 2013.
- [29] J. Calì, D. A. Calian, C. Amati, R. Kleinberger, A. Steed, J. Kautz, and T. Weyrich. 3d-printing of non-assembly, articulated models. *ACM Trans. Graph.*, 31(6):130:1–130:8, Nov. 2012.
- [30] M. Campen, M. Attene, and L. Kobbelt. A practical guide to polygon mesh repairing, 2012. Eurographics Tutorial.

- [31] M. Campen and L. Kobbelt. Exact and robust (self-)intersections for polygonal meshes. *Comput. Graph. Forum*, 2010.
- [32] M. Campen and L. Kobbelt. Polygonal Boundary Evaluation of Minkowski Sums and Swept Volumes. *Comput. Graph. Forum*, 2010.
- [33] CARVE. CARVE: An efficient and robust library for boolean operations on polyhedra, 2014. http://carve-csg.com/.
- [34] CGAL. CGAL, Computational Geometry Algorithms Library, 2015. http://www.cgal.org.
- [35] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], 2015.
- [36] D. Chen, D. I. Levin, P. Didyk, P. Sitthi-Amorn, and W. Matusik. Spec2fab: a reducer-tuner model for translating specifications to 3d prints. ACM Transactions on Graphics (TOG), 32(4):135, 2013.
- [37] X. Chen, H. Zhang, J. Lin, R. Hu, L. Lu, Q. Huang, B. Benes, D. Cohen-Or, and B. Chen. Dapper: Decompose-and-pack for 3d printing. ACM Trans. Graph., 2015.
- [38] A. Cherkaev. Variational methods for structural optimization, volume 140. Springer, 2000.
- [39] S. Choi, Q.-Y. Zhou, S. Miller, and V. Koltun. A large dataset of object scans. arXiv:1602.02481, 2016.
- [40] J. Chu, S. Engelbrecht, G. Graf, and D. W. Rosen. A comparison of synthesis methods for cellular structures with application to additive manufacturing. *Rapid Prototyping Journal*, 16(4):275–283, 2010.

- [41] P. Cignoni, N. Pietroni, L. Malomo, and R. Scopigno. Field-aligned mesh joinery. ACM Trans. Graph., 33(1):11:1–11:12, Feb. 2014.
- [42] D. Cioranescu and P. Donato. An introduction to homogenization. Oxford University Press, 1999.
- [43] T. Davis. Algorithm 832: Umfpack v4. 3—an unsymmetric-pattern multifrontal method. ACM Transactions on Mathematical Software (TOMS), 30(2):196–199, 2004.
- [44] Y. Dong, J. Wang, F. Pellacini, X. Tong, and B. Guo. Fabricating spatially-varying subsurface scattering. ACM Trans. Graph., 29(4):62:1–62:10, July 2010.
- [45] M. Douze, J.-S. Franco, and B. Raffin. QuickCSG: Arbitrary and faster boolean combinations of n solids. Technical Report 01121419, Inria Research Centre Grenoble, Rhone-Alpes, 2015.
- [46] H. Dutagaci, A. Godil, C. P. Cheung, T. Furuya, U. Hillenbrand, and R. Ohbuchi. Shrec'10 track: Range scan retrieval. In *3DOR*, 2010.
- [47] H. Edelsbrunner and E. P. Mücke. Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms. ACM Trans. Graph., 1990.
- [48] I. Elishakof and M. Ohsaki. Optimization and anti-optimization of structures under uncertainty. World Scientific, 2010.
- [49] D. Ewins. Modal testing: theory, practice and application, volume 2. Research studies press Baldock, 2000.
- [50] K. Fidkowski, I. Kroo, K. Willcox, and F. Engelson. Stochastic gust analysis techniques for aircraft conceptual design. 2008. AIAA paper 2008-5848.
- [51] M. Fisher, B. Springborn, P. Schröder, and A. I. Bobenko. An algorithm for the construction of intrinsic delaunay triangulations with applications to digital geometry processing. *Computing*, 81(2-3):199–213, 2007.

- [52] S. Fortune. Vertex-rounding a three-dimensional polyhedral subdivision. Discrete Comput. Geom, 1997.
- [53] Y. Grabovsky and R. V. Kohn. Microstructures minimizing the energy of a two phase elastic composite in two space dimensions. II: the Vigdergauz microstructure. *Journal of the Mechanics and Physics of Solids*, 43(6):949–972, 1995.
- [54] M. Granados, P. Hachenberger, S. Hert, L. Kettner, K. Mehlhorn, and M. Seel. Boolean operations on 3d selective nef complexes: Data structure, algorithms, and implementation. In *Proc. ESA*, 2003.
- [55] G. N. Greaves, A. L. Greer, R. S. Lakes, and T. Rouxel. Poisson's ratio and modern materials. *Nature Materials*, 10(11):823–837, 2011.
- [56] J. K. Guest and J. H. Prévost. Optimizing multifunctional materials: Design of microstructures for maximized stiffness and fluid permeability. *International Journal of Solids and Structures*, 43(2223):7028 – 7047, 2006.
- [57] G. W. Hart. Sculptural forms from hyperbolic tessellations. In Shape Modeling and Applications, 2008. SMI 2008. IEEE International Conference on, pages 155–161.
 IEEE, 2008.
- [58] K. Hauser, C. Shen, and J. O'Brien. Interactive deformation using modal analysis with constraints. In *Graphics Interface*, volume 3, pages 16–17, 2003.
- [59] M. Hašan, M. Fuchs, W. Matusik, H. Pfister, and S. Rusinkiewicz. Physical reproduction of materials with specified subsurface scattering. ACM Trans. Graph., 29:61:1–61:10, July 2010.
- [60] K. Hildebrand, B. Bickel, and M. Alexa. Crdbrd: Shape fabrication by sliding planar slices. *Comp. Graph. Forum*, 31(2pt3):583–592, May 2012.
- [61] J. Hiller and H. Lipson. Design and analysis of digital materials for physical 3d voxel printing. *Rapid Prototyping Journal*, 15(2):137–149, 2009.

- [62] S. J. Hollister. Porous scaffold design for tissue engineering. Nature Materials, 4(7):518–524, 2005.
- [63] W. Hu. Learning 3d object templates by hierarchical quantization of geometry and appearance spaces. In CVPR, 2012.
- [64] A. Jacobson, L. Kavan, and O. Sorkine-Hornung. Robust inside-outside segmentation using generalized winding numbers. ACM Trans. Graph., 2013.
- [65] A. Jacobson, D. Panozzo, et al. libigl: A simple C++ geometry processing library, 2016. http://libigl.github.io/libigl/.
- [66] H. S. Kang. Hierarchical design and simulation of tissue engineering scaffold mechanical, mass transport, and degradation properties. PhD thesis, The University of Michigan, 2010.
- [67] L. Kharevych, P. Mullen, H. Owhadi, and M. Desbrun. Numerical coarsening of inhomogeneous elastic materials. ACM Trans. Graph., 28(3):51:1–51:8, July 2009.
- [68] X. Kou and S. Tan. A systematic approach for integrated computer-aided design and finite element analysis of functionally-graded-material objects. *Materials & design*, 28(10):2549–2565, 2007.
- [69] J. Kruth. Material incress manufacturing by rapid prototyping techniques. {CIRP} Annals - Manufacturing Technology, 40(2):603 – 614, 1991.
- [70] R. Lehoucq, D. Sorensen, and C. Yang. ARPACK users' guide: solution of largescale eigenvalue problems with implicitly restarted Arnoldi methods, volume 6. SIAM, 1998.
- [71] M. Levoy, J. Gerth, B. Curless, and K. Pull. The stanford 3d scanning repository. http://graphics.stanford.edu/data/3Dscanrep/, 2005.
- [72] B. Li, A. Godil, M. Aono, X. Bai, T. Furuya, L. Li, R. J. López-Sastre, H. Johan,
 R. Ohbuchi, C. Redondo-Cabrera, et al. Shrec'12 track: Generic 3d shape retrieval.

In 3DOR, 2012.

- [73] B. Li, Y. Lu, A. Godil, T. Schreck, M. Aono, H. Johan, J. M. Saavedra, and S. Tashiro. Shrec'13 track: large scale sketch-based 3d shape retrieval. In *EW3DOR*, 2013.
- [74] B. Li, Y. Lu, C. Li, A. Godil, T. Schreck, M. Aono, M. Burtscher, H. Fu, T. Furuya,
 H. Johan, et al. Shrec14 track: extended large scale sketch-based 3d shape retrieval.
 In *EW3DOR*, 2014.
- [75] J. J. Lim, H. Pirsiavash, and A. Torralba. Parsing IKEA Objects: Fine Pose Estimation. *ICCV*, 2013.
- [76] C.-Y. Lin, C.-C. Hsiao, P.-Q. Chen, and S. J. Hollister. Interbody fusion cage design using integrated global layout and local microstructure topology optimization. *Spine*, 29(16):1747–1754, 2004. PMID: 15303018.
- [77] C. Y. Lin, N. Kikuchi, and S. J. Hollister. A novel method for biomaterial scaffold internal architecture design to match bone elastic properties with desired porosity. *Journal of Biomechanics*, 37(5):623–636, 2004.
- [78] L. Liu, R. D. James, and P. H. Leo. Periodic inclusionatrix microstructures with constant field inclusions. *Metallurgical and Materials Transactions A*, 38(4):781– 787, 2007.
- [79] L. Luo, I. Baran, S. Rusinkiewicz, and W. Matusik. Chopper: partitioning models into 3d-printable parts. ACM Trans. Graph., 31(6):129:1–129:9, Nov. 2012.
- [80] M. Mahesh, Y. Wong, J. Fuh, and H. Loh. Benchmarking for comparative evaluation of rp systems and processes. *Rapid Prototyping Journal*, 10(2):123–135, 2004.
- [81] MakerBot. Celebrating a maker milestone: 1 million uploads on makerbot's thingiverse . http://www.makerbot.com/blog/2015/10/29, 2015.

- [82] G. Mei and J. C. Tipper. Simple and robust boolean operations for triangulated surfaces. arXiv preprint arXiv:1308.4434, 2013.
- [83] K. Mela and J. Koski. Distributed loads in truss topology optimization. In Proceedings of the 10th world congress on structural and multidisciplinary optimization, Orlando, 2013.
- [84] V. J. Milenkovic and L. R. Nackman. Finding compact coordinate representations for polygons and polyhedra. 1990.
- [85] G. W. Milton. The theory of composites. Cambridge University Press, 2002.
- [86] V. Mironov, R. P. Visconti, V. Kasyanov, G. Forgacs, C. J. Drake, and R. R. Markwald. Organ printing: tissue spheroids as building blocks. *Biomaterials*, 30(12):2164–2174, 2009.
- [87] J. Mitani and H. Suzuki. Making papercraft toys from meshes using strip-based approximate unfolding. In ACM SIGGRAPH 2004 Papers, SIGGRAPH '04, pages 259–263, New York, NY, USA, 2004. ACM, ACM.
- [88] Y. Mori and T. Igarashi. Plushie: An interactive design system for plush toys. In ACM SIGGRAPH 2007 Papers, SIGGRAPH '07, New York, NY, USA, 2007. ACM, ACM.
- [89] A. MOSEK. The mosek optimization tools manual. version 6.0., 2010, 2010.
- [90] S. Mueller, T. Mohr, K. Guenther, J. Frohnhofen, and P. Baudisch. faBrickation: Fast 3d printing of functional objects by integrating construction kit building blocks. In *Proc. SIGCHI*, 2014.
- [91] K. Museth, D. E. Breen, R. T. Whitaker, and A. H. Barr. Level set surface editing operators. ACM Trans. Graph., 2002.
- [92] P. Musialski, T. Auzinger, M. Birsak, M. Wimmer, and L. Kobbelt. Reduced-order shape optimization using offset surfaces. ACM Trans. Graph., 2015.

- [93] A. Myles, N. Pietroni, and D. Zorin. Robust field-aligned global parametrization. ACM Trans. Graph., 2014.
- [94] P. Nair and A. Cavallaro. Shree'08 entry: registration and retrieval of 3d faces using a point distribution model. 2008.
- [95] P. Nakasone and E. Silva. Dynamic design of piezoelectric laminated sensors and actuators using topology optimization. *Journal of Intelligent Material Systems and Structures*, 21(16):1627–1652, 2010.
- [96] P. K. Nathan Silberman, Derek Hoiem and R. Fergus. Indoor segmentation and support inference from rgbd images. In ECCV, 2012.
- [97] B. Naylor, J. Amanatides, and W. Thibault. Merging bsp trees yields polyhedral set operations. In Proc. SIGGRAPH, 1990.
- [98] R. Nickell. Nonlinear dynamics by mode superposition. Computer Methods in Applied Mechanics and Engineering, 7(1):107–129, 1976.
- [99] P. M. Pandey, N. V. Reddy, and S. G. Dhande. Improvement of surface finish by staircase machining in fused deposition modeling. *Journal of Materials Processing Technology*, 132(13):323 – 331, 2003.
- [100] J. Panetta, Q. Zhou, L. Malomo, N. Pietroni, P. Cignoni, and D. Zorin. Elastic textures for additive fabrication. ACM Trans. on Graphics - Siggraph 2015, 34(4):12, aug 2015. Julian Panetta and Quingnan Zhou are Joint first authors.
- [101] D. Pavic, M. Campen, and L. Kobbelt. Hybrid Booleans. Comput. Graph. Forum, 2010.
- [102] A. Pentland and J. Williams. Good vibrations: modal dynamics for graphics and animation. SIGGRAPH Comput. Graph., 23(3):207–214, July 1989.
- [103] D. Pickup, X. Sun, P. L. Rosin, R. Martin, Z. Cheng, Z. Lian, M. Aono, A. B. Hamza, A. Bronstein, M. Bronstein, et al. Shrec14 track: Shape retrieval of non-

rigid 3d human models. In EW3DOR, 2014.

- [104] M. Pratt, A. L. Marsan, V. Kumar, and D. Dutta. An assessment of data requirements and data transfer formats for layered manufacturing. Technical report, National Institute of Standards and Technology, 1998.
- [105] R. Prévost, E. Whiting, S. Lefebvre, and O. Sorkine-Hornung. Make it stand: Balancing shapes for 3d fabrication. ACM Trans. Graph., 2013.
- [106] A. Radman, X. Huang, and Y. Xie. Topological optimization for the design of microstructures of isotropic cellular materials. *Engineering Optimization*, 45(11):1331–1348, 2013.
- [107] L. Sacht, A. Jacobson, D. Panozzo, C. Schüller, and O. Sorkine-Hornung. Consistent volumetric discretizations inside self-intersecting surfaces. *Proc. SGP*, 2013.
- [108] E. Sacks and V. Milenkovic. Robust cascading of operations on polyhedra. Computer-Aided Design (Tech. Note), 2014.
- [109] Y. Schwartzburg and M. Pauly. Fabrication-aware design with intersecting planar pieces. Comput. Graph. Forum, 32(2):317–326, 2013.
- [110] Y. Schwartzburg, R. Testuz, A. Tagliasacchi, and M. Pauly. High-contrast computational caustic design. ACM Trans. Graph., 33(4):74:1–74:11, July 2014.
- [111] J. Schwerdtfeger, F. Wein, G. Leugering, R. F. Singer, C. Krner, M. Stingl, and F. Schury. Design of auxetic structures via mathematical optimization. *Advanced Materials*, 23(22):2650–2654, 2011.
- [112] Shapeways. Things to keep in mind when designing for 3d printing, 2011.
- [113] J. R. Shewchuk. Adaptive precision floating-point arithmetic and fast robust geometric predicates. Discrete & Computational Geometry, 18(3):305–363, 1997.

- [114] P. Shilane, P. Min, M. Kazhdan, and T. Funkhouser. The princeton shape benchmark. In Shape modeling applications, 2004. Proceedings, pages 167–178. IEEE, 2004.
- [115] H. Si. Tetgen: A quality tetrahedral mesh generator and a 3d delaunay triangulator. Weblink: http://tetgen. berlios. de/(accessed on: March 31, 2012), 2007.
- [116] H. Si. A quality tetrahedral mesh generator and a 3D Delaunay triangulator. URL http://tetgen.berlios.de, 2010.
- [117] O. Sigmund. Tailoring materials with prescribed elastic properties. Mechanics of Materials, 20(4):351–368, 1995.
- [118] M. Skouras, B. Thomaszewski, S. Coros, B. Bickel, and M. Gross. Computational design of actuated deformable characters. ACM Transactions on Graphics (TOG), 32(4):82, 2013.
- [119] SolidWorks. Solidworks simulation reference, 2011. http://help.solidworks. com/2011/english/SolidWorks/cworks/LegacyHelp/Simulation/Materials/ Material_models/Linear_Elastic_Orthotropic_Model.htm, 2011.
- [120] O. Stava, J. Vanek, B. Benes, N. Carr, and R. Mĕch. Stress relief: improving structural strength of 3d printable objects. ACM Trans. Graph., 31(4):48:1–48:11, July 2012.
- [121] O. Stava, J. Vanek, B. Benes, N. Carr, and R. Měch. Stress relief: Improving structural strength of 3d printable objects. ACM Trans. Graph., 2012.
- [122] K. Sugihara and M. Iri. Construction of the Voronoi diagram for one million generators in single-precision arithmetic. *Proc. of the IEEE*, 1992.
- [123] A. Telea and A. Jalba. Voxel-based assessment of printability of 3d shapes. In Proceedings of the 10th international conference on Mathematical morphology and its applications to image and signal processing, ISMM'11, pages 393–404, Berlin, Heidelberg, 2011. Springer-Verlag.

- [124] W. C. Thibault and B. F. Naylor. Set operations on polyhedra using binary space partitioning trees. In Proc. SIGGRAPH, 1987.
- [125] S. Timoshenko and S. Woinowsky-Krieger. Theory of plates and shells. Engineering Societies Monographs, New York: McGraw-Hill, 1940.
- [126] S. Torquato. Random heterogeneous materials: microstructure and macroscopic properties, volume 16. Springer, 2002.
- [127] S. Torquato and A. Donev. Minimal surfaces and multifunctionality. Proceedings of the Royal Society of London. Series A: Mathematical, Physical and Engineering Sciences, 460(2047):1849–1856, 2004.
- [128] S. Torquato, S. Hyun, and A. Donev. Multifunctional composites: optimizing microstructures for simultaneous transport of heat and electricity. *Physical review letters*, 89(26):266601, 2002.
- [129] S. Torquato, S. Hyun, and A. Donev. Optimal design of manufacturable threedimensional composites with multifunctional characteristics. *Journal of Applied Physics*, 94(9):5748–5755, 2003.
- [130] G. Turk. The stanford bunny http://www.cc.gatech.edu/~turk/bunny/bunny.html, 2000.
- [131] N. Umetani, T. Igarashi, and N. J. Mitra. Guided exploration of physically valid shapes for furniture design. ACM Trans. Graph., 31(4):86:1–86:11, July 2012.
- [132] N. Umetani and R. Schmidt. Cross-sectional structural analysis for 3d printing optimization. In SIGAsia Technical Briefs, 2013.
- [133] G. Varadhan, S. Krishnan, T. Sriram, and D. Manocha. Topology preserving surface extraction using adaptive subdivision. In *Proc. SGP*, 2004.
- [134] R. C. Veltkamp, S. van Jole, H. Drira, B. B. Amor, M. Daoudi, H. Li, L. Chen,
 P. Claes, D. Smeets, J. Hermans, et al. Shrec'11 track: 3d face models retrieval.

In 3DOR, 2011.

- [135] K. Vidimče, S.-P. Wang, J. Ragan-Kelley, and W. Matusik. Openfab: A programmable pipeline for multi-material fabrication. ACM Transactions on Graphics (TOG), 32(4):136, 2013.
- [136] C. C. L. Wang. Approximate boolean operations on large polyhedral solids with partial mesh reconstruction. *IEEE TVCG*, 2011.
- [137] T. Weyrich, P. Peers, W. Matusik, and S. Rusinkiewicz. Fabricating microgeometry for custom surface reflectance. ACM Trans. on Graphics (Proc. SIGGRAPH), 28(3):32:1–32:6, 2009.
- [138] K. V. Wong and A. Hernandez. A review of additive manufacturing. ISRN Mechanical Engineering, 2012, 2012.
- [139] S. Xu and J. Keyser. Fast and robust booleans on polyhedra. Computer-Aided Design, 2013.
- [140] Z CORPORATION. Architectural design guide printing 3d architectural models, 2011.
- [141] T. Zeiler. Matched filter concept and maximum gust loads. Journal of aircraft, 34(1):101–108, 1997.
- [142] H. Zhao, C. C. Wang, Y. Chen, and X. Jin. Parallel and efficient boolean on polygonal solids. *The Visual Computer*, 2011.
- [143] Q. Zhou. A study in fabricating microstructures (part 1), 2015.
- [144] Q. Zhou, E. Grinspun, D. Zorin, and A. Jacobson. Mesh arrangements for solid geometry. ACM Transactions on Graphics (TOG), 35(4), 2016.
- [145] Q. Zhou, J. Panetta, and D. Zorin. Worst-case structural analysis. ACM Trans. Graph., 2013.