Computational Design for the Next Manufacturing Revolution

by

Adriana Schulz

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Doctor of Philosophy in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2018

© Massachusetts Institute of Technology 2018. All rights reserved.

Author . . . .
Department of Electrical Engineering and Computer Science
May 22, 2018

Certified by. . . . .
Wojciech Matusik
Associate Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Leslie A. Kolodziejski
Professor of Electrical Engineering and Computer Science
Chair, Department Committee on Graduate Students

To my great-grandmother, who dreamt about an academic life.
To my parents, who empowered me to live it.

Computational Design for the Next Manufacturing Revolution

by

Adriana Schulz

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2018, in partial fulfillment of the
requirements for the degree of
Doctor of Philosophy in Computer Science and Engineering

Abstract

Over the next few decades, we are going to transition to a new economy where highly complex, customizable products are manufactured on demand by flexible robotic systems. In many fields, this shift has already begun. 3D printers are revolutionizing production of metal parts in the aerospace, automotive, and medical industries. Whole-garment knitting machines allow automated production of complex apparel and shoes. Manufacturing electronics on flexible substrates makes it possible to build a whole new range of products for consumer electronics and medical diagnostics. Collaborative robots, such as Baxter from Rethink Robotics, allow flexible and automated assembly of complex objects. Overall, these new machines enable batch-one manufacturing of products that have unprecedented complexity.

In this thesis, I argue that the field of computational design is essential for the next revolution in manufacturing. To build increasingly functional, complex and integrated products, we need to create design tools that allow their users to efficiently explore high-dimensional design spaces by optimizing over a set of performance objectives that can be measured only by expensive computations. In this work discuss how to overcome these challenges by 1) developing data-driven methods for efficient exploration of these large spaces and 2) performance-driven algorithms for automated design optimization based on high-level functional specifications. I showcase how these two concepts are applied by developing new systems for designing robots, drones, and furniture.

Thesis Supervisor: Wojciech Matusik
Title: Associate Professor of Electrical Engineering and Computer Science

Acknowledgments

First and foremost, I would like to thank my PhD advisor, Wojciech Matusik, for his many teachings on conducting research: from picking good problems to framing and presenting ideas, and all that takes place in between. Wojciech always ensured I was granted exposure to the important tools and means that would help me grow into a successful researcher. Most of all, I thank him for encouraging me to dare to do things I would never have known I was capable of. This was only possible because he challenged me and made it safe for me to try.

I would also like to thank my other committee members, Daniela Rus and Eitan Grinspun, for their teachings and mentoring. I thank Eitan for nurturing my interest in geometry, for teaching me how to defend my ideas so that I could better understand what makes a research problem interesting, and for all the wonderful advice over hummus at Columbia Hillel. I thank Daniela for introducing me to the amazing field of robotics, for showing me how to bring together ideas from different fields, and for always encouraging me to âĂIJkeep the gradient!”.

I would also like to thank the wonderful collaborators without whom this work would not be possible: Ariel Shamir, Justin Solomon, Ilya Baran, David Levin, Pitchaya Sitthi-amorn, Changxi Zheng, Bo Zhu, Andrew Spielberg, Tao Du, Jie Xu, Jeffrey Lipton, and Bernd Bickel. I am extremely grateful not only for the ideas and work they have contributed to this thesis, but also for the fun memories we created—especially during the weeks leading up to the Siggraph deadlines! I also thank the brilliant students I had the pleasure to supervise and who greatly contributed to this work: Wei Zhao, Harrison Wang, Robin Cheng, Baker Logan, Keneth Pinera, Luis Trueba, Katie Bartel, Jacob Haip, Nilu Zhao, Saul Lopez, Jackson Wirekoh, Molly Donalson, Helena Wang, Kendall Helbert, Alexxis Isaac, Isaque Dutra, Megan Chao, Diane Rosales, Marie Moudio, Brian Saavedra, and Emily Salvador.

I would like to thank the rest of the MIT faculty, students, and staff who provided great feedback, support, and friendship during this process. I thank David Karger, Fredo Durand, Solar-Lezama, Sylvain Paris, Boris Katz, and Charles K. Smart for the helpful ideas and discussions. I thank all the students from the MIT Graphics Group—especially Valentina Shin, James Minor, Desai Chen, Abe Davis, Javier Ramos, Michael Foshey, and Nick Bandeira—as well as the students at MIT DRL and Columbia Computer Graphics Group. I thank Bryt Bradley for being an incredible supporter over all these years.

I also want to thank the wonderful friends I made during the last few years as I was pursuing this degree. They made this process fun, made the hard moments better, and inspired me and changed me in more ways than I can describe. It would be impossible to name them all, but I'd like to especially thank Josh Pfeffer (without whose support I cannot imagine having submitted half of my papers), Natalie Mashian Fisher, and Luis Voloch.

Next, I would like to acknowledge and thank my mentors from my undergraduate and Masters studies in Brazil, Luiz Velho and Eduardo da Silva. Though they were not directly involved with the work in this thesis, they were fundamental in preparing me for it. I am very grateful for their teaching, mentorship, and support during my initial ventures in the academic world.

Last but not least, I want to thank my family. I thank my parents, Sonia and Mauro Schulz, who always nurtured my curiosity, who empowered me to come to MIT, and who supported me in every step of the way. I thank them for their constant love, closeness,

and for sending me far away so that I could pursue my own dreams. I know how hard that was and still is. I thank them for their contagious passion for academia, for being amazing role models, and for always pushing me to go a step beyond. I thank my mom for the long philosophical conversations, for introducing me to art and design, and for always challenging me to think deeply and outside the box. I thank my dad for inspiring my love for mathematics and for encouraging me to study Engineering, all the while pointing out to me how limited logic actually is. Finally, I thank my sister, Lilian Schulz, my lifelong best friend. My sister was always there to help me out—from my homework in grade school to my faculty applications—and none of my achievements would have been half as fun if she had not been there to celebrate them with me!

A Story Before We Start

In contrast to the rest of this thesis, where I have verified all the finding to the best of my abilities, this story may not be entirely true. But I will take the liberty to share it, because there is something personal about a thesis that makes it different from my list of published papers. If this work is supposed to summarize what I have discovered during my PhD, then this story, however accurate or inaccurate it may be, is essential to making this work complete.

They say Rousseau loved the opera so much that he would visit this specific opera house once every week. And he would sit in the back listening to the women's voices—such marvelous, enchanting voices, that he would imagine them to be the most beautiful women in the world. One day, a friend, aware of his adoration, agreed to arrange for him to have dinner with the singers one evening after the show.

On the day the dinner was scheduled, Rousseau came to the opera house delighted. He sat that night in his usual seat in the back, completely captivated by the singers and waiting in anticipation to meet the women to whom such voices belonged. But as he joined the dinner party he was surprised to see women who were very far from the beauty standards he envisioned. Most of the women had birth deformities or had suffered physical traumas that had left them disfigured. Because of their unattractive exterior, they had been taken care of by an orphanage that sheltered them and educated them to pursue a career where only their voices could be heard. Rousseau was so surprised, it took him a moment to gather himself and join the party. But the women were sweet and cheerful, so the dinner was pleasant and Rousseau came home content.

A week passed by and once again Rousseau made his way to the opera house. From his usual seat in the back he heard the women's voices. They were as graceful and dazzling as they has always been. He closed his eyes and imaged how women who possessed such voices looked. He quickly realized he had seen their faces a week before. But as he sat and heard their voices, their angelic, bewitching voices, he imaged them to be the most beautiful women in the world.

The person who started this PhD, imagined academia to be "the most beautiful place in the world." An open invitation to push even if ever so slightly the invisible wall that surrounds us—the frontier of human knowledge. The one who writes this thesis has seen academia in a much brighter light. There were surprises that were not particularly pleasant. There were times I needed a few moments to "gather myself." But I can still see that invisible wall and now I am surrounded by incredible friends and mentors who are actively pushing it. It has been a privilege to witness their work, their intellectual integrity, their passion, their bad hair days. Academia enchants me today even more than it ever did.

The main discovery of my PhD was this imperfect yet amazing world, which I hope to always be a part of.

Contents

1 Introduction 29
1.1 The Next-Manufacturing Revolution 29
1.2 Abstraction of Design for Manufacturing 30
1.3 Approach and Contributions 32
1.3.1 Data-Driven Methods 32
1.3.2 Performance-Driven Search Methods 33
1.3.3 End-to-End Systems 34
1.4 Thesis Overview 35
1.5 Contributions 35

2 Related Work 37
2.1 Data-Driven Methods 37
2.1.1 Parametric Modeling 38
2.1.2 Parametric CAD 38
2.2 Performance-Driven Methods 40
2.2.1 Interactive Exploration 40
2.2.2 Optimization 40

3 A Collection of Manufacturable Designs 43
3.1 Introduction 43
3.2 Manufacturable Designs 43
3.2.1 Items Catalog 43
3.2.2 Set of Designs 44
3.3 Parametric Manufacturable Designs 45
3.4 Automatic Hierarchical Parametrization 45
3.5 Defining the mapping function F 46
3.5.1 Geosemantic Relationships 47
3.6 Connections 49
3.7 Discussion 50

4 Retrieval on Collections of Manufacturable Designs 51
4.1 Introduction 51
4.2 Related Work 53
4.3 Representation of Parametric Shapes 55
4.3.1 Manifold Approximation 56

4.4 Algorithm . . . 57
4.4.1 Unbounded Manifolds . . . 58
4.4.2 Bounded Manifolds . . . 60
4.5 Retrieval . . . 62
4.6 Experimental Setup . . . 63
4.6.1 Database . . . 63
4.6.2 Descriptors . . . 64
4.7 Evaluation . . . 65
4.7.1 Manifold Representation . . . 65
4.7.2 Retrieval . . . 69
4.7.3 Limitations . . . 73
4.8 Discussion . . . 75

5 Assembly-Based Design for Manufacturing 77
5.1 Introduction . . . 77
5.2 Design Workflow . . . 78
5.3 Parametric Manipulations . . . 79
5.4 Composition . . . 81
5.5 Snapping . . . 81
5.6 Connecting . . . 83
5.6.1 Searching for Connections . . . 83
5.6.2 Final Composition . . . 84
5.7 Results . . . 85
5.7.1 Modeling . . . 85
5.7.2 Fabrication . . . 86
5.8 Discussion . . . 86

6 Interactive Design-Space Exploration 89
6.1 Introduction . . . 89
6.2 Related Work . . . 90
6.3 Workflow . . . 91
6.4 Precomputation Overview and Notations . . . 91
6.4.1 Refinement Relations . . . 93
6.4.2 Adaptive Sampling . . . 94
6.4.3 Refinement Notations . . . 94
6.5 Adaptive Refinement Strategy . . . 95
6.5.1 Motivation . . . 95
6.5.2 Algorithm . . . 98
6.5.3 Extension to Cubic B-splines . . . 101
6.6 Homeomorphic Mapping . . . 103
6.6.1 Motivation . . . 103
6.6.2 Algorithm . . . 104
6.7 Results . . . 107
6.7.1 Application in Shape Optimization . . . 110
6.7.2 Limitations . . . 111

6.8 Discussion . . . 112

7 **Interactive Performance-Space Exploration** **113**
7.1 Introduction . . . 113
7.2 Related Work . . . 114
7.3 Mathematical Preliminaries . . . 115
    7.3.1 Definitions . . . 115
    7.3.2 KKT Conditions . . . 116
7.4 First-Order Approximation . . . 117
7.5 Pareto Front Discovery . . . 118
    7.5.1 Data Structure . . . 118
    7.5.2 Discovery Algorithm . . . 119
    7.5.3 First-Order Approximation . . . 121
    7.5.4 Sparse Approximation . . . 122
    7.5.5 Visualization . . . 123
7.6 Results . . . 123
    7.6.1 Experiments . . . 124
    7.6.2 Design Applications . . . 128
7.7 Discussion . . . 131

8 **Applications** **133**
8.1 Introduction . . . 133
8.2 Interactive Design of Ground Robots . . . 134
    8.2.1 System Overview . . . 134
    8.2.2 Methods Overview . . . 136
    8.2.3 Results . . . 138
8.3 Interactive Multicopter Design . . . 142
    8.3.1 System Overview . . . 142
    8.3.2 Methods Overview . . . 142
    8.3.3 Results . . . 143
8.4 Robot-Assisted Carpentry . . . 145
    8.4.1 Systems Overview . . . 146
    8.4.2 Methods Overview . . . 146
    8.4.3 Results . . . 148

9 **Conclusion** **151**
9.1 Future Work . . . 151
    9.1.1 Data-Driven Methods . . . 152
    9.1.2 Performance-Driven Methods . . . 153
    9.1.3 End-to-End Systems . . . 155
9.2 Lessons Learned . . . 156

A Proofs of Interpolation Algorithm 157
- A.1 Notation 157
- A.2 Properties of Step 2 157
- A.3 Local Point Lemma 158
- A.4 Locality proof with Linear Precision 159
  - A.4.1 Preservation over Basis Refinement 159
  - A.4.2 Preservation over Element Refinement 160
- A.5 Example 164

B Proof of the First-Order Approximation of the Pareto Front 167

List of Figures

1-1 The next manufacturing revolution. In a very near future, we will have workshops where automated manufacturing machines will make customized, complex, multi-material parts and collaborative robots will allow flexible and automatic assembly of objects that have multiple functionalities. 30

1-2 Designing a robot means selecting a point in a high-dimensional *design space* that includes specification of the robot's geometry, electronic components, and control software. Users select this point based on how it is mapped to a *performance space*, which defines important metrics that designers take into account when optimizing the models (e.g., the trajectory, fabrication cost, and reaction to an external force). 31

1-3 An example of a parametric design. The original design is shown in gray, and the new designs generated by varying the parameters are shown in yellow. . 33

1-4 We apply data-driven and performance-driven algorithms to develop interactive systems for design of complex functional mechanisms. From left to right: a data-driven method that automatically handles manufacturing details and guarantees [Schulz et al., 2014]; real-time performance exploration and optimization for functionality-driven design [Schulz et al., 2017c]; end-to-end system for design and manufacturing of ground robots that allows for concurrent design of both geometry and motion [Schulz et al., 2017b]. . . . 34

2-1 When engineers design shapes, they embed their experience and knowledge in carefully selected, fabrication-aware parameters such as this *fillet* radius, which encapsulates a curved transition along an edge, and this *chamfer* distance, which describes the slope transition between two surfaces. These include fabrication limitations that take into account the different processes. This example includes minimum radius constraints on internal cuts for compatibility with milling machines. 39

2-2 Regeneration times for parameter changes using Onshape for models with increasing level of complexity. 40

3-1 An example of a fabricable object from our collection (left). Each design is detailed down to the level of individual screws, and each part maintains a reference to the items used from the items catalog (right). 44

3-2 From left to right: a design example of a toy wagon, the hierarchical tree, and a visualization of the connections. The arrow on the handle indicates that this part has an articulation, namely that it can rotate along the depicted axis. The tree includes the geosemantic relationships that are stored at each level of the hierarchy, C_0 to C_4 (shown in blue), as well as connections (depicted in red). The visualization on the right illustrates the information contained in each connection node. 46
3-3 A parametric model with pattern elements. Upon resizing, both the number of floor planks and the number of rungs in the monkey bars change. 47
3-4 We show a simple 2D table consisting of three parts, a top and two legs (Leg1 and Leg2). Each part is contained within a single element for which the $\mathbf{q}^i$ are the positions (x, y) and sizes $(\Delta x, \Delta y)$ of the bounding box of the part. . 48

4-1 We propose a method for shape retrieval from parametric shape collections that uses a descriptor space representation. While shape descriptors map single shapes to points in a descriptor space, smooth descriptors map parametric shapes to low dimensional manifolds in this space. Our method efficiently represents these manifolds in order to allow for accurate and fast retrieval of the closest parametric model to a given query shape. 52
4-2 The function $\mathcal{M}(q) = (\mathcal{D} \circ \mathcal{F})(q)$ is a composition of the mapping function $\mathcal{F}$ from parameter values to a geometry with the signature function $\mathcal{D}$ which generates a descriptor for a given geometry. 56
4-3 The coverage of a point (left) and of a tangent line (right) are defined by the region of the manifold (here illustrated as a curve $c(t)$) that is well approximated by this primitive given the allowed approximation error δ. While the coverage of the point $c(0)$ is directly proportional to δ, the coverage of the tangent line $l(t)$ is proportional to d, which depends on the curvature. 58
4-4 Computation of the bounding radius for a tangent space primitive $l(t)$ on the manifold $c(t)$. In the illustration, the dotted line represents the part outside the boundary of the manifold and δ is the allowed approximation error. Left: when we do not take the boundary into account the bounding radius is determined uniquely by the curvature constraint r_c. Right: when we are close to the boundary, the radius is computed as $r_b + r_\delta$, where r_b is the distance to the boundary and r_δ is the amount by which we can expand the radius preserving tightness constraints. We can compute r_δ from δ and d_b, which is the distance from the boundary point p_b on $l(t)$ to the manifold. . . 60
4-5 From left to right: covering the manifold with tangent spaces bounded by hyperspheres, non-oriented ellipsoids, and oriented ellipsoids. This example illustrates that the number of primitives needed to represent the manifold for the same value of δ is reduced when we use better primitives. We notice that even in this example with a 2-dimensional parameter space there is a significant improvement when oriented primitives are used. The blue dots represent the underlying manifold represented via super sampling. (Please note that these are high dimensional primitives projected to 2D for visualization and therefore appear slightly distorted.) 62

4-6 Approximating the distance d_e from the projected point p to the hyperellipse. Let S be a scaling function that maps the hyperellipse to the unit hypersphere centered at the origin. The point on the hypersphere that is the closest to $S(p)$ is given by $p_{contact} = \frac{S(p)}{\|S(p)\|}$. We use the inverse mapping and approximate the distance from p to the hyperellipse as $d_e \approx \|p - S^{-1}(p_{contact})\|$. 63

4-7 Comparison between adaptive sampling and rejection sampling on a simple paraboloid example. Our rejection sampling scheme was done for both point and planes for a fixed approximation error δ. The number of samples for the adaptive sampling schemes was chosen to be the same as the result of the rejection sampling for both points and planes. The top row shows the results for point samples. Though both methods return a uniform distribution, in the adaptive sampling scheme points tend to clump together and leave gaps. The bottom row results for approximating with tangent spaces (we only display the center of the tangent space for simplification). Once again both methods display the desired distribution (based on curvature) and rejection sampling covers the space more effectively. 66

4-8 Measuring *fitting* and *coverage* errors as a function of the target parameter δ for the implemented descriptors. we observe that both measured errors are within the bounds of δ. For large values of δ we observe that the fitting error drops to zero. This is because for very coarse approximations, our algorithm prefers points to tangents – the coverage of points become larger with δ, while plane coverage is still limited by the curvature and the boundary of the manifold. Since absolute distance values depend on descriptors (and are much larger for the Light Field descriptor), the ranges of the target errors for this experiment were chosen so that the number of samples were similar for all descriptors. 67

4-9 Comparison between our hybrid method and using a single primitive. Top row: shows the storage cost of each representation across different target parameters δ in log scale. Bottom row: the relative cost of the single primitive methods while compared to our method. 68

4-10 Demonstration of query failures when representation only consists of the mean shape. From left to right: the mean shape of some parametric models in the database, query shape given by a random parameter setting of each parametric model, and the closest mean shape retrieved from the database. Since changes in parameter settings significantly alter the geometry, the closest mean shapes are usually not from the parametric models that originate the queries. . . . 70

4-11 Comparison between our approach and the naive one. We measure the difference between the distance to the closest primitive in the collection and the distance to the correct manifold and show the worst case for both approaches for querying points sampled on the full database and on individual categories. From these results we verify that our method has a better performance across all categories. 71

4-12 Results of retrieval. From left to right: query shape, result for D2 descriptor (for increasing target errors), and result for Voxel descriptor (for increasing target errors) and results for the Light Fied descriptor (for increasing target errors). 71

4-13 Precision-recall plots evaluating classification accuracy for our method compared to using only mean shapes for different descriptors. 73

4-14 Comparison of retrieval with mean shapes only and manifold representation for the Light Field descriptor. From left to right: query shape (green), closest mean shape retrieved (gray), closest parametric shape retrieved with parameter fitting (blue) with its corresponding mean shape (gray). We observe that using the parametric shapes we retrieve models that are more similar in geometry but may lie on a different class. 74

5-1 The design and fabrication by example pipeline: casual users design new models by composing parts from a database of parametric manufacturable designs. The system assists the users in this task by automatically aligning parts and assigning appropriate connectors. The output of the system is a detailed model that includes all components necessary for fabrication. 77

5-2 The user interface. Icons that link to components of the database are displayed on the left, and the modeling canvas is on the right. 79

5-3 An illustration of how parametric variations can be explored in our tool. The arrows control translation, while spheres control scaling. On the left, we show the controls on a leaf node of the hierarchy; on the right, we show the controls on an internal node. During manipulation, elements on the selected node are represented in full color, while the others become semi-transparent. Notice that constrained degrees of freedom are hidden. For example, the user is unable to change the thickness of the shelf, since the items catalog states that planks of wood can be cut only in two directions. 80

5-4 An example of snapping to constraints. We add a tabletop T^A to the working model T^W containing eight legs (right). The coplanarity constraints on the original design T^D that contained T^A are represented by the normals of the corresponding planes (left; we show only the vertical ones). The feasible snapping configurations for $\mathbf{q}^A$ are shown on the right. The system will choose one of these configurations: its choice will depend on the scale parameters and the position on which the user places the tabletop. 82

5-5 An illustration of snapping for functional objects. When a door is added to the side of a cabinet, it automaticaly rescales so that, when shut, it will align with the oposite side. At left, a door is added to the working model. From left to right: the added door before snapping, the snapped configuration, and a visualization of the snapped configuration when the door is closed. The rotation axis of the articulations is depicted by the arrows. 83

5-6 An example of changing parameters to fit connectors. From left to right: the bottom shelf snapped to the bottom of the table, the resulting configuration of the model after the connecting step, and the vizualization of the connectors (principal elements are made semi-transparent). Notice that, in order to connect the bottom shelf to the table legs, the system raises the shelf above the ground to leave room for l-brackets. 85
5-7 Examples of models designed using our system and the number of individual parts they comprise. Different colors indicate the different parts that were added to the model. 85
5-8 An example of different manipulations of a working model after it has been composed from multiple parametric designs. 86
5-9 From left to right: input designs, models created using the system, and fabricated results. We highlight the connecting elements on the first model by making all principal elements semi-transparent. 87

6-1 Our method takes as input a CAD model with a set of exposed parameters which define the design space. We sample the parametric space in an adaptive grid and propose techniques to smoothly interpolate this data. We show how this can be used to drive interactive exploration tools that allow designers to visualize the shape space while geometry and physical properties are updated in real time. 90
6-2 The neighborhood of an element, denoted $\mathcal{B}(e_l)$, is defined as the set of adjacent samples (left). The neighborhood of a *sample*, denoted $\mathcal{N}(x_k)$, is defined as the set of adjacent elements (middle). We also extend the definition of $\mathcal{N}(x_k)$ to any point $p \in \mathcal{A}$ as the set of adjacent elements (right). 92
6-3 When locality is enforced, the number of consistent representations p_k^l that are needed at each sample x_k depends on the cardinality of $\mathcal{N}(x_k)$. 93
6-4 Refinement of linear B-splines. 93
6-5 A linear B-spline ϕ_i^j is illustrated in blue: the blue "‘x"’ is the center and the region where it assumes non-zero values is shaded in light blue. The local sample set of this linear B-spline $\mathcal{L}(\phi_i^j)$, is defined as the set of samples x_k such that $S(\phi_i^j) \subset \mathcal{N}(x_k)$ and is illustrated in black. 95
6-6 Comparison between multi-linear interpolation (left), where discontinuities appear along T-junctions, and our method (right), where the interpolation is continuous. 96
6-7 Our method of combining element refinement with basis refinement: interpolated values inside each element only depend on the samples that lie on the boundary of that element. 96
6-8 Hierarchical and quasi-hierarchical basis refinement strategies. Both of these schemes violate locality on element e_{1A} since the basis function ϕ_1^0 does not vanish on this element. The sample outside $\mathcal{B}(e_{1A})$ affecting this element is highlighted in red. 97
6-9 Example of samples added for a given split in 2D and 3D. Split element, split plane and added samples are shown in blue. 98

6-10 Illustration of refinement in two dimensions. Letters are used to index x_k for clarity. When the element is split, new samples are added, the B-splines ϕ_i^j are refined and the new basis functions ψ_i^j are recomputed to ensure locality. 100
6-11 The neighborhood $\overline{\mathcal{N}}(x_k)$ of a sample x_k (left) and the neighborhood $\overline{\mathcal{B}}(e_l)$ of an element e_l (right) for cubic B-splines. 101
6-12 Refinement of cubic B-splines. 102
6-13 The mapping $F : p_{\bar{k}}^{\bar{l}} \to p_{\bar{k}}^{l}$ is used to obtain p_k^l for $x_k \in \overline{\mathcal{B}}(e_l) \setminus \mathcal{B}(e_l)$. 103
6-14 Example of referencing in Onshape. The user applies a feature to a given face which is highlighted (left), and then changes parameter making other faces and edges of the model merge or split (right). Onshape's referencing scheme guarantees that the feature will still be applied to the correct face even after these changes are made. 104
6-15 Two examples of a fillet feature applied to edges. After this feature is defined, parameter changes on earlier features in the feature list split the edges into two parts. The way the edge references are handled in order to re-generate the fillet depends on the feature history, not the geometry. On the model on the left, the CAD system applies the fillet to both edges generated by the split; on the model on the right, the fillet is applied to only one of the edges. 105
6-16 Common patch layout given a source and target geometry. Top to bottom: B-rep with highlighted topological changes, paths extracted from CAD, complete patch layout. 106
6-17 Mapping result. Given a source geometry and a target mesh topology the mapping outputs a mesh with the source geometry and target topology. . . . 106
6-18 Examples of interactive visualization. On the left we show each model in red boxes and the regions with fixed boundary conditions and forces with red arrows. On the right we show results from our visualization interface. As the user varying the parameters, geometry is updated is real time. The top four rows show pre-computed stress analysis. On the forth row the force direction also varies and is illustrated with a red arrow. The fifth row shows a result with fluid simulation. The last row shows a results with thermoelastic simulation where the colors display (from left to right) heat distribution, deformation and stress. 108
6-19 Result of shape optimization. Stress is minimized on an arbor press for two different use cases, a tall one is shown on the right and a short one on the left. 111

7-1 Our method allows users to optimize designs based on a set of performance metrics. Given a design space and a set of performance evaluation functions, our method automatically extracts the *Pareto set*—those design points with optimal trade-offs. We represent Pareto points in design and performance space with a set of corresponding manifolds (left). The Pareto-optimal solutions are then embedded to allow interactive exploration of performance trade-offs (right). The mapping from manifolds in performance space back to design space allows designers to explore performance trade-offs interactively while visualizing the corresponding geometry and gaining an understanding of a model's underlying properties. 114

7-2 The Pareto set represents the points in design space with optimal performance trade-offs that get mapped to the Pareto front in performance space. Different colors indicate different manifolds in design and performance space with a one-to-one mapping. Any ray from the origin (blue line) can only intersect the Pareto front once. 115

7-3 The performance buffer: since a ray from the origin can only intersect one point in the Pareto front, we use a buffer discretized by (hyper)spherical coordinates for storage. 119

7-4 A single iteration of the discovery algorithm: random samples $\mathbf{x}_s^i$ are generated from the current data on the buffer (illustrated in gray) and optimized for a search direction $\mathbf{s}(\mathbf{x}_s^i)$ (blue arrow). A first-order approximation around the result of this optimization, $\mathbf{x}_o^i$, generates the corresponding manifolds in both design and performance space (red lines) and the buffer is updated based on this new data. 120

7-5 Search directions for buffer cells for $d = 2$. For diversity, different regions of the performance space get assigned different search directions. We use the buffer discretization to define these directions as illustrated in the figure. . . 121

7-6 Nondominated solutions for various well-established benchmark problems using our proposed approach. Top two rows: Solutions for the five real-valued ZDT problems. Bottom row: Solutions for the first three DTLZ problems with three objectives. Our approach was able to converge to the ground truth Pareto front in all cases. 124

7-7 Results for Pareto front discovery on Fourier benchmark. The figure illustrates how the Pareto front can be covered by a set of individually smooth regions which are mapped via F from affine subspaces in the design space. Each corresponding pair in design and performance space is illustrated in a different color. In high-dimensional cases in design space, dimensionality-reduction via principal component analysis is applied for visualization purposes. 126

7-8 A direct piecewise-linear interpolation result of the example shown on the first row of Figure 7-7. The optimal solution is chosen from the list of points at each buffer cell (blue points) and a denser sampling is generated by linearly interpolating the preimage of neighboring points in performance space (red points). Since the Pareto front is comprised of distinct manifolds, the linearly interpolated points have no guarantee of Pareto optimality. For illustrative purposes, the interpolation is performed on a sparse set of the discovered solutions. 127

7-9 Performance Space Assessment of the approximation error associated with our first-order expansion method on the example shown on the first row of Figure 7-7. Left: plot in performance space of the discovered Pareto front (grey) and results of the same points after performing an additional local optimization (maroon). Right: histogram showing the approximation error for points on the discovered Pareto front. 127

7-10 First-order approximation for a point $\mathbf{x}$ on the Pareto front of the bi-objective Kursawe problem Kursawe [1991] (true front shown in black). The first-order approximation defined by our method (red) is compared to the mapping of affine spaces around $\mathbf{x}$ generated using directions chosen uniformly at random (gray). 128

7-11 Examples of CAD models processed using our proposed technique. From left to right: Pareto-optimal points in design space (illustrated with multi-dimensional scaling via principal component analysis for models with more than three design parameters); Pareto-optimal points in performance space; the resulting embedding and illustrations of geometric results for some sampled points. Across all figures, the different colors correspond to different regions resulting from local expansion in design space. 129

7-12 Example of two variations of the brake hub that have similar performance metrics but very different design parameters. Our system can expose these relationships providing intuition to designers about the structural nature of the model. The metrics are shown under a normalization for easy comparison—the Pareto front is rescaled to lie between zero and one. 130

7-13 Example of two different lamps (front and side view) that have the same performance across all metrics. Due to the large dimensionality of the design space, these two configurations have the same stability, mass, and distance to the focal point. 131

8-1 A robot design that topples while walking (a) can be modified to follow a gait that only wobbles slightly (b), but changing the geometry (c) allows the robot to move much faster and more steadily. 134

8-2 System diagram. Users interact with geometry and gait design tools. The designed models are simulated to provide feedback to the user. The user may iterate over the design before fabricating and assembling the model. 135

8-3 User interface. Icons that link to geometry components are displayed on the left and the gait design tool is on the bottom. Users design models by dragging components into the center canvas and editing them. Performance metrics for the design are shown on the right. 135

8-4 Geometry components by category. Our system's database contains 45 components: 12 bodies, 23 limbs, and 10 peripherals. 136

8-5 Joint controllers for each joint type. Controllers for every joint are separated into a step phase (shown in green) and a reset phase (shown in red). Users modify gaits by changing the θ_i values for each joint and defining the step sequence. 137

8-6 One of the geometry components in the database, with both its 3D shape and 2D unfolding. The component has parameter values of diameter q_d and width q_w. The 3D and 2D representations are coupled so that they are simultaneously updated as parameters are changed. In order to allow the assembly-based modeling to work with our fabrication method, each component is also annotated with a set of connecting patches that indicate where components are allowed to be connected together. The green line indicates a patch for functional connections, and the blue lines are patches for static connections. 137

8-7 Our system automatically generates a 3D mesh for a foldable design. (a) The original design, with blue edges indicating static connections and green dots indicating functional connections. (b) Connections are processed by shrinking faces to make room for hinges and adding holes for servo mounts. (c) Hinges are added and the result is a complete mesh that is 3D printed. 138

8-8 The system provides guidance arrows for users who want to make local geometry optimizations. The up and down arrows indicate that the user should lengthen and shorten the leg, respectively. No arrow indicates that the part dimension is already at a local optimum. 138

8-9 Optimization results for a four-legged fish robot for different metrics. Maximizing speed increases the overall size of the robot (b). Minimizing wobble makes the robot shorter (c). 139

8-10 Cars designed by eight different novice users after a 20 min. training session with the tool. Users were given 10 min. to design their car. 139

8-11 Gallery of designs created by a novice user after a 20 min. training session. Each of the models took between three and 25 min. to design and contains multiple components from the database. 140

8-12 Trajectories designed by users during the second task. Users who were allowed to make changes to the robot geometry were able to make the robot navigate the course about 40% faster on average. 140

8-13 Six robots designed and fabricated using the system. 141

8-14 The electronics (servomotors, microcontroller, and battery) inside the Monkey example (Figure 8-13(d)). 141

8-15 Comparison of monkey gait in simulation and from physical robot. The robot motion matches those from the simulations. 142

8-16 An overview of our system. The pipeline consists of multicopter design, optimization, simulation, fabrication, and flight tests. 143

8-17 Example of composition. Left: parts with highlighted patches (circular patch with an annotated main axis and a diameter in blue and flat patch with an annotated normal in orange). Right: composed design. 143

8-18 Pentacopter pairs. Left: original pentacopter design. Right: optimized pentacopter for larger payload. 144

8-19 Left: motor outputs of the unoptimized pentacopter with 1047g payload. Motor 1 and 3 reach saturation point (PWM=1800) at 23s. Increasing the payload will cause them to saturate constantly and therefore fail to balance the torques from other motors. Note that motor 5 is not fully exploited in this copter. Right: motor outputs of the optimized pentacopter with 1392g payload. Motor 2 and 4 reach saturation during the flight. Compared with the unoptimized pentacopter, all five motors are now well balanced, making it possible to take over 30% more payload. 144

8-20 Optimizing a quadcopter with flight time metric and geometry constraints. Left: a standard quadcopter. Right: optimized rectangular quadcopter. . . . 145

8-21 Battery change when a quadcopter hovers. Left: battery voltage. Given the same amount of time the optimized quadcopter ends up having a larger voltage. Right: battery current. In steady state the optimized copter requires less current. 145

8-22 System workflow. Experts design parametric models in a commercial CAD system. End users customize and verify the designs using our interactive interface. Once the users are satisfied with the model, the system outputs a complete fabrication plan which includes instructions for robot assisted cutting processes and rules for user assembly. 146

8-23 Expert input using CAD software. Experts create a model in a standard CAD system which is parametric from construction and exposes the parameters for user customization using the system's variables (top). For defining the assembly, experts use a custom *connection* feature which takes in two parts, a face for connection and the priority of the connection for assembly (bottom). The experts use this feature to define all of the connections on the model and those are used by our algorithm to automatically generate assembly instructions.147

8-24 User customization: the design tab (top) and the stimulation tab that display the stress distribution (middle) and elastic deformations (bottom). 147

8-25 Chop saw process from left to right: Robots team lift the lumber, transport it, place it on the chop saw to re-grasp, slide the lumber to the proper length, and the chop saw cuts the lumber. 148

8-26 Sequence of assembly steps shown in the composition interface. The user can traverse the list of steps, select parts to view additional information, and use the 3D window to view the connections from different viewpoints. 148

8-27 Stress distribution on different variations of the chair model (top) and elastic deformation on variations on the shed model (bottom). 149

8-28 Variations of the deck model. From left to right: input terrain, deck model instance visualized on the terrain, stress distribution, elastic deformation. . . 150

8-29 Fabrication of the table model. From left to right: eight pieces of stock 1x3 lumber, the cutting of the pieces, the final parts, and the assembled table result. 150

A-1 Centers of linear B-splines at different levels. The set $\mathcal{I}^0$ consists only of the centers denoted by circles, the set $\mathcal{I}^1$ includes the centers denoted by circles and those marked by diamonds, and $\mathcal{I}^2$ includes all the centers denoted in the figure. 158

A-2 Contrasting the interpolation solution from our method with the interpolation solution adding virtual nodes. Top row: the interpolation solution from our method. The resulting interpolation is equivalent to a uniform basis function at the coarsest level j such that every sample on the element boundary belongs to $\mathcal{I}^j$ (in this case $j = 1$). The values q_i^j can be computed hierarchically at points for which samples p_i^j do not exist. In this example, q_3^1 is based on the average of the adjacent samples $x_0, x_6 \in \mathcal{I}^0$, and q_4^1 is the average of the four corner samples, also contained in $\mathcal{I}^0$. Bottom row: interpolation solution adding virtual nodes. The color display on the right illustrates how our method restricts the impact of a sample in $\mathcal{I}^j$ to $\mathbf{s}_j$. 164

List of Tables

4.1 Parametric designs in our collection. 64
4.2 Comparison between coverage regions in descriptor space. 72

6.1 The number of parameters (K), levels of the K-d tree, total sampled nodes, average mesh size (in number of tetrahedrons), and total storage (in number of stored meshes) for each model. 109
6.2 Relative approximation error on geometry and elastic FEM for all example models. 110

8.1 Pentacopter specifications. Motor angle is the angle between motor orientation and up direction. 145
8.2 Fabrication information for the models in Figure 8-27: number of parts that will be processed with the jig saw and chop saw, total number of used pegs, and number of assembly steps. 149

Chapter 1

Introduction

1.1 The Next-Manufacturing Revolution

Novel technologies indicate that manufacturing is about to undergo an unprecedented and radical shift. The key revolutionary aspects of new fabrication devices are:

- **Complete Automation.** New manufacturing devices can handle end-to-end fabrication of complete products. This is in contrast to prior automation of short and repetitive tasks that could only handle part of the manufacturing process, making it necessary to have long production lines to incrementally build a single product.

- **Versatility.** New fabrication devices can execute not only a single task, but can be used to produce a variety of different results depending on the input instructions. This allows for the fast and inexpensive fabrication of products that are customized to meet specific needs.

- **High Product Complexity.** New technologies have significantly increased the complexity, performance, and functionality of final products. For example, by allowing materials to be specified at incredibly high precision and resolution, novel manufacturing platforms enable the production of new objects with unseen mechanical properties.

Overall, these new technologies indicate that over the next few decades we are going to transition to a new economy where highly complex, customizable products are manufactured on demand. In many fields, this shift has already begun. 3D printers are revolutionizing production of metal parts in the aerospace, automotive, and medical industries. Whole-garment knitting machines allow automated production of complex apparel and shoes. Manufacturing electronics on flexible substrates makes it possible to build a whole new range of consumer electronics and medical diagnostic devices. Collaborative robots, such as Baxter from Rethink Robotics, allow flexible and automated assembly of complex objects.

From the perspective of designers and engineers, these advances can enable: 1) increased freedom in defining the form and material properties of objects, 2) the possibility of integrating multiple functional components into a single product, and 3) easy access to manufacturing

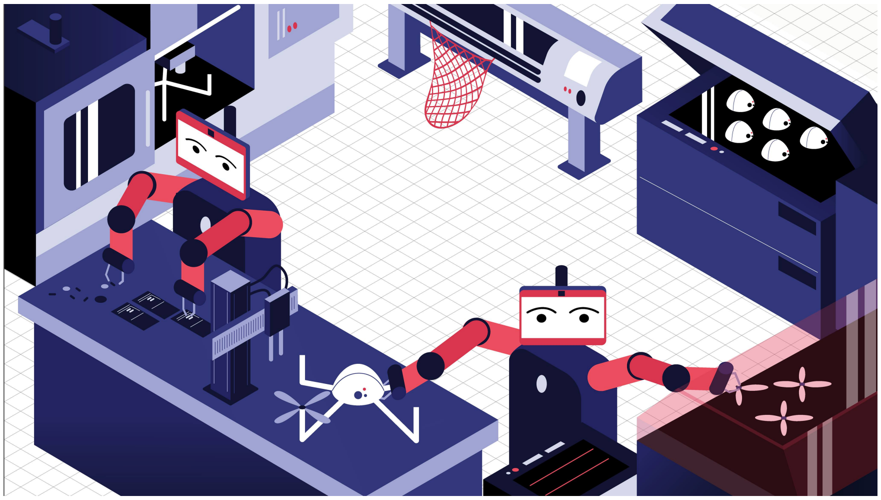


Figure 1-1: The next manufacturing revolution. In a very near future, we will have workshops where automated manufacturing machines will make customized, complex, multi-material parts and collaborative robots will allow flexible and automatic assembly of objects that have multiple functionalities.

devices, even if they have little expertise. Therefore, while past revolutions in manufacturing have been mostly focused on productivity, these new advances have the potential to fundamentally change *what* can be manufactured and by *whom*.

Figure 1-1 shows what these future workshops may look like. These "‘mini-factories"’ will include a whole range of automated manufacturing machines, from laser cutters to 3D printers and knitting machines. These devices will allow fast fabrication of high performing parts, which will then be assembled by collaborative robots into complex functional objects. The easy integration of multiple complex parts into single objects will allow for products with multiple functionalities. In addition, the versatility allowed by these machines will enable on-demand manufacturing, bringing about a new age of personalization and customization.

This shift, however, cannot happen through innovations in *hardware* alone. To build increasingly functional, complex, and integrated products, we also need to create *design software* that allows users to efficiently explore a design space that incorporates geometry, materials, electronics, and fabrication processes. Such design tools must be equipped to simultaneously optimize many different objectives, from the aesthetics of the product to its durability and motion. Further, to empower more people to create their own personalized objects, we also need to develop tools that are more accessible to users with little design experience.

The central question that drives this thesis is: *what should the future design tools for manufacturing be?*

1.2 Abstraction of Design for Manufacturing

In this work, we will address this question using the abstraction shown in Figure 1-2 to define design from manufacturing. This figure illustrates the problem of designing a robot for ground locomotion, which involves specifying the robots geometry, the materials, the electronics, and the control software. Together, these attributes define a very high dimensional *design space*. Under this abstraction, to design means to select a point in this space.

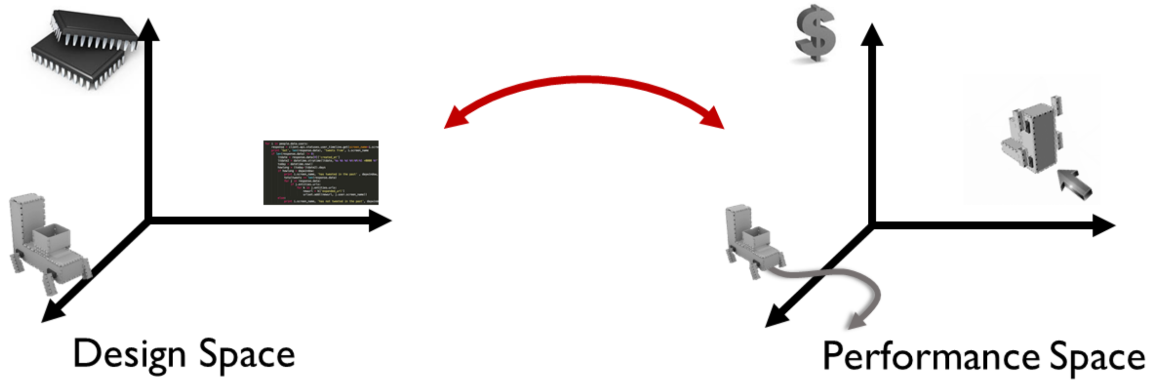


Figure 1-2: Designing a robot means selecting a point in a high-dimensional *design space* that includes specification of the robot's geometry, electronic components, and control software. Users select this point based on how it is mapped to a *performance space*, which defines important metrics that designers take into account when optimizing the models (e.g., the trajectory, fabrication cost, and reaction to an external force).

But how do we select this point? Design of functional objects depends on a set of performance metrics that determine how these objects will behave once they are physically realized. In the case of the ground robot, examples of relevant performance metrics are: locomotion speed, reactions to external forces, and the overall fabrication cost. As shown in Figure 1-2, the set of performance metrics define the *performance space*.

Under this abstraction, design means to select a point in *design space* based on how it maps to the *performance space*. This can be done by directly exploring the design space driven by performance feedback (*forward* problem), or by solving an *inverse* problem: finding the point in design space that optimizes user-specified performance objectives.

In this thesis, we will discuss algorithms and interactive systems that guide users in solving both the forward and the inverse problem. The main challenges in devising such tools are twofold:

- **Complex Design Spaces.** Typical design spaces are not only high dimensional, they are also sparse—many combinations of materials and geometry are impossible to manufacture, and many combinations of electronic components are useless. The complexity of this domain makes search and optimization a challenge.
- **Complex Performance Evaluations.** Typical performance evaluations can only be measured by expensive computations since they require predicting physical behavior. In addition, performance metrics are often conflicting—increasing one will decrease the other. Algorithms should guide designers and engineers in navigating a complex landscape of compromises, generating objects that perhaps do not optimize any single quality measure, but rather are considered optimal under a given performance trade-off.

In this work, we address these challenges to propose methods for efficient exploration of high-dimensional design spaces and algorithms for automated design optimization based on functional specifications.

1.3 Approach and Contributions

Our approach is to combine data-driven strategies with efficient search methods. Data-driven strategies attempt to discover certain attributes of the underlying design space in order to estimate performance. While fast, they provide limited insight into a model's underlying structure and can be difficult to debug. On the other hand, search-based strategies attempt to fully explore the design space with combinatorial search or continuous optimization algorithms. While these methods tend to be more robust and effective in finding solutions that achieve a desired performance, they are prohibitively slow in typical high-dimensional designs spaces.

In this work, we combine these two strategies to develop new techniques that can effectively discover optimized designs and are also fast enough to be used in interactive tools. Our approach is to first to constrain the design spaces in meaningful ways using *data-driven methods*. This can be done either using hand-crafted rules or deriving these rules from data (machine learning). We then propose new *performance-driven search methods* in these reduced spaces. These methods allow users to either explore the design space with real-time performance feedback or to solve inverse design problems.

In summary, the central thesis of this dissertation can be expressed in the following statement:

> *Efficient design tools for functional mechanisms can be developed by constraining design spaces using data-driven methods and then defining efficient performance-driven search algorithms on these reduced spaces.*

1.3.1 Data-Driven Methods

The data-driven methods we present constrain the design spaces to regions with manufacturability guarantees. By automatically handling manufacturing details, this approach ensures that every model designed with the tool can be physically realized. It also allows users to more intuitively and efficiently explore the design space by allowing them to focus on high-level conceptual design. Finally, this approach can make design more accessible to users who do not have the expertise to specify manufacturing details.

To define such data-driven strategies, our work leverages collections of manufacturable *parametric* designs: consistent families of manufacturable models, each of which is described by a particular point in parameter space. The parametric model makes computations more efficient by both reducing the search space and constraining manipulations to structure-preserving and manufacturing-aware variations (see Figure 1-3 for an example of a parametric design).

An important problem that arises in data-driven design is how to enable efficient searching in these types of collections. This is challenging because these collections are partially discrete

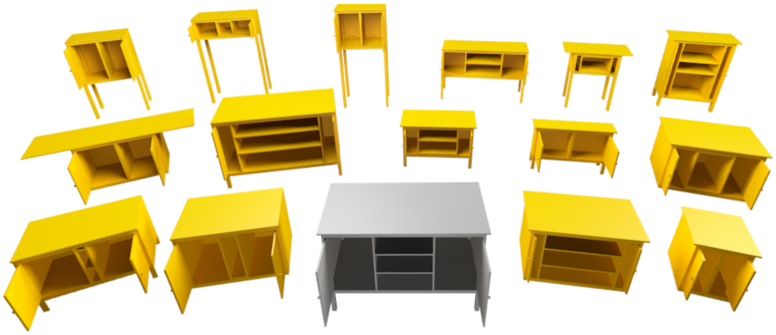


Figure 1-3: An example of a parametric design. The original design is shown in gray, and the new designs generated by varying the parameters are shown in yellow.

(number of shapes) and partially continuous (parameter values). To address this, we have proposed a descriptor-based approach for accurate shape-based matching and retrieval on parametric shape collections [Schulz et al., 2017c].

We have also used these collections in an interactive design system to create new functional manufacturable models in a design-by-example manner [Schulz et al., 2014]. In this system, a simple interface allows novice users to create new designs by combining parts from the database. Data-driven algorithms automatically handle connections and manufacturability constraints (see Figure 1-4).

1.3.2 Performance-Driven Search Methods

With the data-driven methods we propose, users can focus on high-level design without worrying whether the results can be physically realized. However, users need to consider not only whether an object can be manufactured, but also how an object will perform once built. Performance-driven design techniques are crucial for typical applications in which the design goals are dictated by how the finished products will function in the physical world. However, it can be quite difficult to incorporate such techniques into interactive design tools.

As discussed above, the first main challenge is that evaluating performance typically requires time-consuming simulations. We address this challenge by proposing an algorithm that allows real-time performance evaluation and optimization over reduced design spaces defined by parametric models (see Figure 1-4). To achieve interactive rates, we use precomputation on an adaptively sampled grid and propose a scheme for interpolating in this domain, where each sample is a mesh with different combinatorics. Our solution defines a novel interpolation algorithm for adaptive grids that is both continuous/smooth and local [Schulz et al., 2017c]. This allows interactive exploration of combined design and performance space.

The other main challenge is that design for manufacturing applications typically requires simultaneous optimization of conflicting performance objectives: design variations that improve one performance metric may decrease another performance metric. In these scenarios there is no unique optimal design, but rather a set of designs that are optimal for different trade-offs (called Pareto-optimal). Using the fact that the set of solutions with optimal performance trade-offs lie on the boundary of the projection of the design space onto the performance space (the *Pareto* front), we propose a new method to efficiently represent this domain for real-time exploration. Our approach is based on a first-order approximation of the Pareto front derived from duality theory in multi-objective optimization [Schulz et al., 2018]. This allows real-time exploration of joint performance and design space for different trade-offs.

1.3.3 End-to-End Systems

We show the capabilities of our approach by applying data-driven and performance-driven algorithms to develop interactive *end-to-end* systems for designing complex functional mechanisms. One of the important considerations in building such systems is handling concurrent design across multiple domains (e.g., geometry, materials, electronics, software, and fabrication processes), which must all be considered when designing complex functional mechanisms.

We have proposed an end-to-end tool for designing and manufacturing robots with ground locomotion [Schulz et al., 2017b]. To enable the concurrent design of a robotâĂŹs shape and motion, we leverage a data collection that includes parametric gaits and geometric robot parts, and we couple it with algorithms for interactive simulation feedback. The system outputs a complete fabrication plan and assembly instructions for the robotâĂŹs mechanical body, electronic components, and control software (see Figure 1-4). We also extend these ideas to the design of multicopters, simultaneously designing their geometry and control to optimize their flying capabilities [Du et al., 2016].

Finally, we develop a system for mass customization of carpentry items for robot-assisted manufacturing [Lipton et al., 2018]. This system uses parametric expert data to define a space of customizable designs that can be fabricated with a proposed robotics system. A simple interface allows casual users to explore the defined design space with performance feedback.

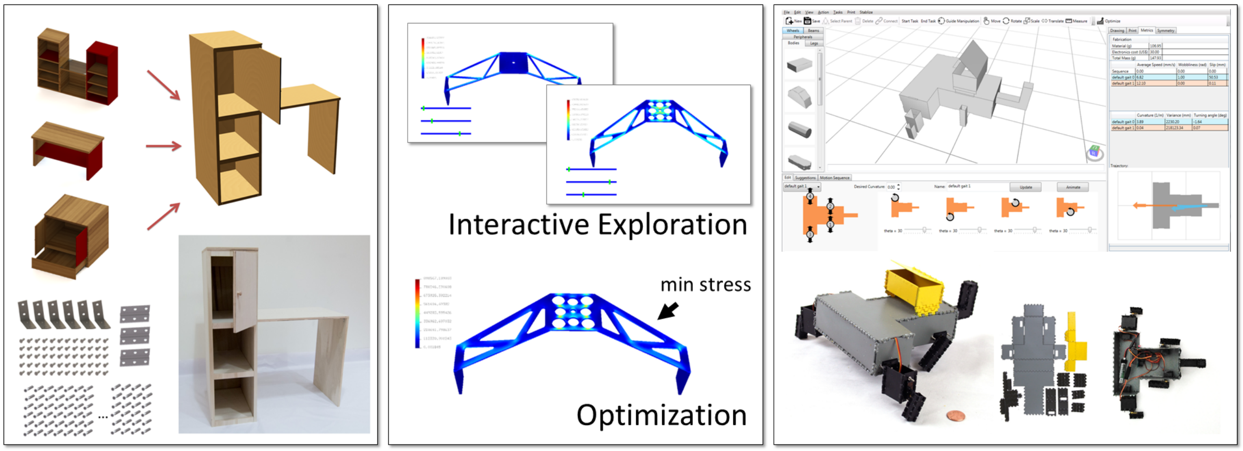


Figure 1-4: We apply data-driven and performance-driven algorithms to develop interactive systems for design of complex functional mechanisms. From left to right: a data-driven method that automatically handles manufacturing details and guarantees [Schulz et al., 2014]; real-time performance exploration and optimization for functionality-driven design [Schulz et al., 2017c]; end-to-end system for design and manufacturing of ground robots that allows for concurrent design of both geometry and motion [Schulz et al., 2017b].

1.4 Thesis Overview

The rest of this thesis is organized as follows.

In the second chapter we present background material and describe the most significant previous work in computational design for manufacturing.

Chapters 3-5 are dedicated to data-driven design techniques which allow constraining the design space in meaningful ways based on parametrization. In Chapter 3, we describe a parametric collection of manufacturable designs which we have made publicly available at *http://fabbyexample.csail.mit.edu*. In Chapter 4, we present an algorithm for efficient shape-based matching and retrieval in such parametric shape collections. In Chapter 5, we outline a fabrication-aware composition algorithm which allows users to create new designs by mixing and matching database components.

We then proceed to describe performance-driven techniques. The sixth chapter is dedicated to forward methods, where we propose an algorithm for interactive exploration of the design space with real-time performance feedback. Inverse methods are discussed in the seventh chapter, where we describe a multi-objective optimization approach for exploration of the performance space with real-time feedback on the corresponding design.

In the final technical chapter of this thesis, we present end-to-end design systems which incorporate both data-driven and performance-driven algorithms. We first present a tool for design of robots with ground locomotion that allows simultaneous design and analysis of geometry and motion. We then describe a system for design of multicopters, which require concurrent optimization of geometry and control. Finally, we show a system for carpentry design and mass customization with robotic-assisted manufacturing.

We conclude the thesis in Chapter 9, summarizing the results and discussing directions for future work.

1.5 Contributions

In summary, this thesis makes the following contributions:

- A database of parametric manufacturable designs that can be used for computational fabrication (Chapter 3, [Schulz et al., 2014]).
- An algorithm for retrieval on parametric shape collections (Chapter 4, [Schulz et al., 2017a]).
- A data-driven method for design-by-composition that ensures manufacturability (Chapter 5,[Schulz et al., 2014]).
- A method for interactive performance-driven design space exploration (Chapter 6, [Schulz et al., 2017c]).
- A general method for interpolation on K-d trees that guarantees locality and continuity (Chapter 6, [Schulz et al., 2017c]).
- A method for interactive exploration of performance trade-offs (Chapter 7, [Schulz et al., 2018]).

- A Pareto front discovery algorithm based on a first order approximation derived from duality theory in multi-objective optimization (Chapter 7,, [Schulz et al., 2018]).
- An interactive system for design of ground robots that allows for concurrent design of shape and motion. (Chapter 8, [Schulz et al., 2017b]).
- An interactive system for design of multicopters that allows for concurrent optimization of shape and motion (Chapter 8, [Du et al., 2016]).
- An interactive system for design of carpentry items for robot-assisted manufacturing (Chapter 8, [Lipton et al., 2018]).

Chapter 2

Related Work

Manufacturability has long been a concern in engineering design Currier [1980], Gupta and Nau [1995], Khardekar et al. [2006], Patel and Campbell [2010], Wang and Sturges [1996] and more recently has become an area of interest in the computer graphics community. Proposed fabrication-oriented systems include tools for plush toys Mori and Igarashi [2007], furniture Lau et al. [2011], Saul et al. [2011], Umetani et al. [2012], clothes McCann et al. [2016], Umetani et al. [2011], inflatable structures Skouras et al. [2014], wire meshes Garg et al. [2014], model airplanes Umetani et al. [2014], twisty puzzles Sun and Zheng [2015], architecture-scale objects Yoshida et al. [2015], mechanical objects Coros et al. [2013], Koo et al. [2014], Thomaszewski et al. [2014], Zhu et al. [2012], and electromechanical systems Bezzo et al. [2015], Megaro et al. [2015b], Mehta et al. [2015], Song et al. [2015].

In this work, we approach computational design from the abstraction described in Figure 1-2—computational tools should guide users in searching the design space to find a solution that best satisfies performance specifications. To address the challenges highlighted in the previous chapter, we proposed methods that combine data-driven approaches and performance-driven algorithms. In what follows we will discuss relevant related work in each of these areas.

2.1 Data-Driven Methods

Shape collections have been widely used to allow data-driven geometric modeling. Modeling by example [Funkhouser et al., 2004] enables the design of new models by combining parts of different shapes from a large database. More recent work focuses on data-driven suggestions for adding details [Chaudhuri and Koltun, 2010] and modeling [Chaudhuri et al., 2011]. Similarly, recombination of model parts has been used to expand databases [Jain et al., 2012, Kalogerakis et al., 2012] and repair low-quality 3D models [Shen et al., 2012b]. In data-driven methods, production rules are implied by the dataset and they do not have to be explicitly distilled.

Nevertheless, none of these works explore the fabrication aspect of data-driven modeling. Creating models that can be physically realized adds a number of challenges to the modeling pipeline that are addressed in this work. We propose using collections of manufacturable design—models that have all the information necessary for fabrication. In order to ensure manufacturability is preserved during manipulation and composition, we take advantage of parametric modeling.

2.1.1 Parametric Modeling

Parametric models are consistent families of geometries, each described by a given point in parameter space. Parametric shapes are advantageous because they constrain the manipulations to structure-preserving (and potentially fabrication-aware) variations and also reduce the search space, making computations more efficient. These techniques have been applied to many types of parametric shapes, from mechanical objects to garments, and masonry. Recent usage of parametric shapes for modeling includes reconstruction of 3D shapes from images Xu et al. [2011] or point-set scans [Nan et al., 2012, Shen et al., 2012a]. Talton et al. [2011] use a parametric grammar for procedural modeling.

Previous work in computational design has exploited different techniques for parametric representations. A common approach is to convert a single input mesh into a parametric model using methods such as cages [Zheng et al., 2011b], linear blend skinning [Jacobson et al., 2011] or manifold harmonics [Musialski et al., 2015]. These methods generate smooth deformations which have been shown to work on shapes that are more organic and for fabrication methods such as 3D printing [Prévost et al., 2013] or water-jet cutting of smooth 2D parametric curves [Bharaj et al., 2015]. For more mechanical structures, cages techniques have been applied together with algebraic techniques to preserve discrete regular patterns [Bokeloh et al., 2012b].

These, however, have been limited to very small datasets with highly constrained variations (such as dimensions of wooden plates or carbon tubes) that require laborious annotations from mechanical engineers. Because of these limitations, some recent work looked at CAD as a means to incorporate meaningful, fabrication-aware, parametric variations [Koyama et al., 2015, Shugrina et al., 2015]

2.1.2 Parametric CAD

Practically every man-made shape that exists has once been designed by a mechanical engineer using a parametric CAD tool (e.g., SolidWorks, OpenScad, Creo, Onshape), and these models are available on extensive repositories (e.g., GrabCAD and Thingiverse, Onshape's public database). CAD shapes have the advantage of being parametric from construction, and the parameters exposed by expert engineers contain specific design intent, including manufacturing constraints (see Figure 2-1). Parametric CAD allows for many independent variables, but these are typically constrained for structure preserving consideration, the need to interface with other parts, and manufacturing limitations. In this context, engineers typically expose a small number of variables which can be used to optimize the shape.

Early CAD systems were based on constructive solid geometry (CSG), which defines hierarchical boolean operations on solid primitives. Modern professional CAD systems, however, use B-rep methods that start from 2D sketches and model solids with operations such as extruding, sweeping, and revolving [Stroud, 2006]. A B-rep is a collection of surface elements in the boundary of the solid and is composed of a topology (faces, edges, vertices) and a geometry (surfaces, curves, points). Each topological entity has an associated geometry

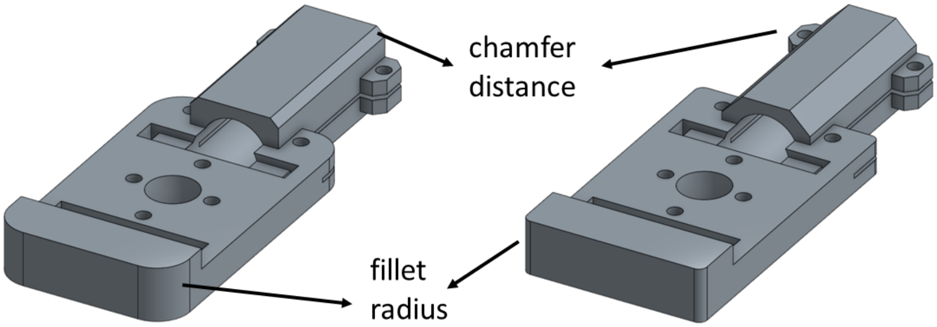


Figure 2-1: When engineers design shapes, they embed their experience and knowledge in carefully selected, fabrication-aware parameters such as this *fillet* radius, which encapsulates a curved transition along an edge, and this *chamfer* distance, which describes the slope transition between two surfaces. These include fabrication limitations that take into account the different processes. This example includes minimum radius constraints on internal cuts for compatibility with milling machines.

(e.g., a face is a bounded portion of a surface). Modern CAD systems use the *feature-based* modeling paradigm [Bidarra and Bronsvoort, 2000], in which solids are expressed as a list of features that define instructions on how the shape is created. These systems are also *parametric*: each feature depends on a set of numerical values or geometric relationships that control the shape of the models [Farin et al., 2002].

The geometric operations in each feature (e.g., sweeping, booleans, filleting, drafting, offsetting) are executed by geometric kernels (such as ACIS or Parasolid) that are often licensed from other companies. An important aspect of feature execution that is handled internally by CAD systems is the *referencing* scheme. This scheme uniquely identifies the parts of the geometry to which each feature is applied. This mechanism is what allows models to re-generate appropriately as users make changes in parameters in the feature history [Baba-Ali et al., 2009, Bidarra et al., 2005].

Historically, however, it has not been a relevant concern of CAD systems to perform real-time exploration on a set of exposed parameters. In CAD systems, shape generation from a parameter configuration is done by recomputing the list of operations that construct the geometry. This involves computationally expensive operations (e.g., sweeping, booleans, filleting, drafting, offsetting). This means that every time a designer tunes CAD parameters of a complex model, it can take on the order of minutes for the system to recompute the geometry (see Figure 2-2).

In this work we use parametrization to constrain this design space ensuring manufacturability (Chapters 3). In order to harness collections of CAD desins, we propose a novel algorithm from real-time evaluation of CAD geometry from user input parameters (Chapter 6).

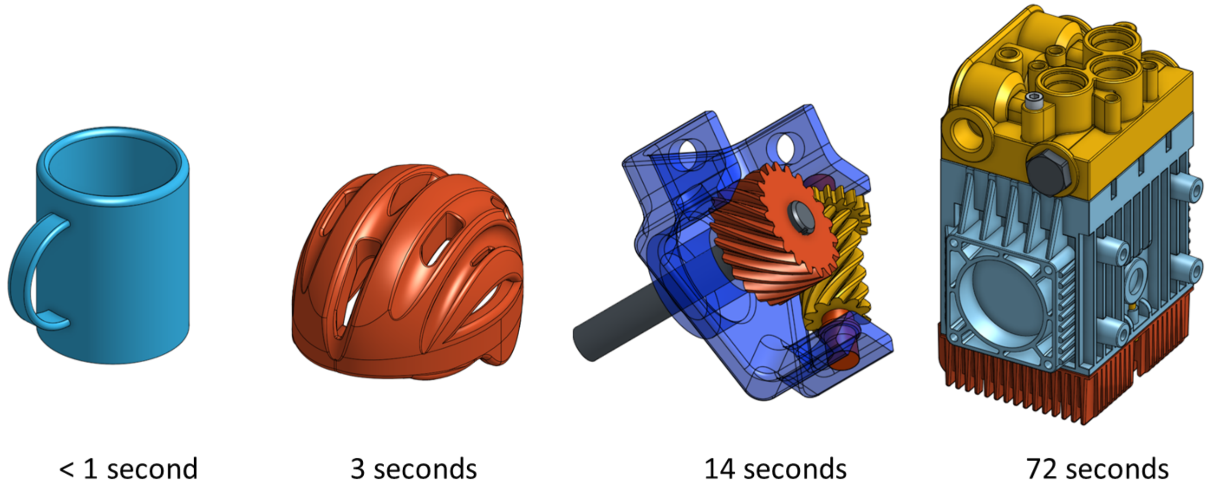


Figure 2-2: Regeneration times for parameter changes using Onshape for models with increasing level of complexity.

2.2 Performance-Driven Methods

2.2.1 Interactive Exploration

After using data-driven methods to constrain the design space in meaningful ways, we search the space driven by performance objectives. Search methods aim at guiding users in finding shape variations that meet certain design goals. There are two main approaches for this task. The first is to use interactive systems that allow users to manually explore the design space. These techniques are chosen in order to give more control to the users or because design objectives depend on multiple considerations. Recent works in this space include furniture design [Umetani et al., 2012], interlocking structures [Skouras et al., 2015], model airplanes [Umetani et al., 2014], garment design [Umetani et al., 2011], and robots [Megaro et al., 2015a], to name a few.

In our work, we extend these ideas to the space of parametric CAD models with performance objectives that can only be measured by expensive computations. To allow interactive exploration with real-time performance feedback in these generic domains, we describe a precomputation-based approach driven by a novel interpolation scheme (Chapter 6).

2.2.2 Optimization

An alternative to interactive exploration is direct shape optimization. This technique requires less user input and is preferred when objective functions can be easily exposed. Example of recent work on optimization include structural stability of masonry buildings [Whiting et al., 2012] or 3D printed shapes [Prévost et al., 2013], frequency spectra [Bharaj et al., 2015], structure and motion of linkages [Bächer et al., 2015], and buoyancy [Musialski et al., 2015, 2016]. Optimization has the advantage of being automatic while user exploration can be time-consuming.

A fundamental limitation of typical design optimization techniques is that they require a single objective function for evaluating performance. In most applications, however, multiple criteria are used to evaluate the quality of a design. Structures must be stable *and* lightweight. Vehicles must be aerodynamic, durable, *and* inexpensive to produce. In most cases, the performance objectives are not only multiple but also *conflicting*: Improving a design on one axis often decreases its quality on another axis. In reality, designers and engineers navigate a complex landscape of compromises, generating objects that perhaps do not optimize any single quality measure but rather are considered optimal under a given performance trade-off.

In such scenarios, it is challenging, if not impossible, to determine a single objective function for direct optimization. Therefore, an exploration interface can more effectively guide users to solutions with desired performance trade-offs. Our approach is to use offline optimization to reduce the search space to solutions that achieve optimal performance trade-offs. We then define an exploration interface that allows users to search in this subset of the *performance* space. This approach lets us handle problems with multiple performance objectives while reducing the amount of user effort during exploration (Chapter 7).

The set of solutions with optimal performance trade-off are called *Pareto*-optimal in multi-objective optimization literature. Pareto-optimal points correspond to solutions where it is impossible to improve on one performance metric without making at least one other metric worse. While multi-objective optimization approaches that discover a set of Pareto-optimal points have become increasingly common in engineering practice [Agrawal and van de Panne, 2013, Bandaru and Deb, 2015, Deb and Srinivasan, 2006], in this work we propose a novel technique that finds a sparse representation of the full Pareto front, allowing for interactive exploration of design trade-offs.

Our approach is also related to works on material design which use pre-computation to find the gamut of achievable material properties [Bickel et al., 2010, Dong et al., 2010, Zhu et al., 2017]. In these works, the design space is the set of materials that can be output by a given device and the performance metrics are the material properties that are evaluated. The fundamental difference is that instead of a gamut which defines all the possible material combinations in a d-dimensional material property space, our application requires defining a combination of $(d-1)$-dimensional manifolds that represents the optimal performance trade-offs.

Chapter 3

A Collection of Manufacturable Designs

3.1 Introduction

Creating a collection of manufacturable designs is the first step in proposing data-driven design for manufacturing techniques. While many collections of man-made shapes are publicly available (e.g. GrabCAD), such repositories of 3D models do not contain the necessary information for fabrication. We have therefore created a new collection of fabricable models with the help of design experts—in this case, a group of mechanical engineers. Each design is modeled using a commercial CAD software and is composed of a collection of standard parts that can be purchased from suppliers. It also includes all assembly details, such as connectors and support structures, ensuring the object is manufacturable.

To build a database that can be leveraged for data-driven design tools, we propose a hierarchical parametric representation that includes connectivity constraints between parts. We automatically convert these specific designs to parameterized models by extracting constraints and parameters from the models. This allows us to perform structure-preserving manipulations using both discrete and continuous parameters of the parts. The data is available at *http://fabbyexample.csail.mit.edu*.

3.2 Manufacturable Designs

The data is divided into two sections: an items catalog containing a list of commercial items, and a set of *designs* constructed by domain experts. Each part used in a design references a corresponding item in the catalog. This imbues the data with the unique property that all the designs can actually be manufactured.

3.2.1 Items Catalog

An *item* is a physical part that can be purchased from a variety of suppliers or manufacturers. The items catalog lists the available items along with all information required for their use during the design and fabrication process: their corresponding material type (e.g., wood, metal, glass), their price, their dimensions, and a 3D mesh representing their geometry.

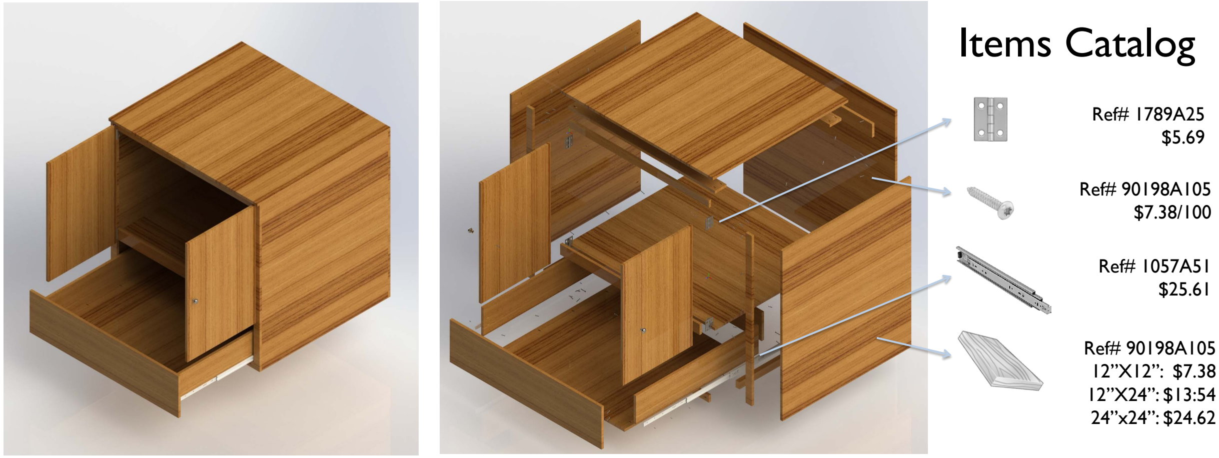


Figure 3-1: An example of a fabricable object from our collection (left). Each design is detailed down to the level of individual screws, and each part maintains a reference to the items used from the items catalog (right).

The items within the catalog incorporate two additional pieces of information that are used in later stages of our method. First, each dimension of an item is labelled as either fixed or resizable. We only allow resizing of items if a corresponding physical process is possible. For example, we allow resizing of wooden components because they may be cut using available tools. This information is used during the design parameterization process. Second, each item maintains links to external suppliers (e.g., McMaster-Carr), allowing for easy sourcing during fabrication.

3.2.2 Set of Designs

The design set contains a large number of manufacturable models (henceforth called *designs*) created using commercial CAD software (Solidworks). Each design is an assembly of parts, and the parts all contain links to the corresponding items in the items catalog (Figure 3-1). The parts are grouped into subassemblies in a hierarchical fashion, as is common in standard CAD tools. Once a design has been finalized, we build an associated connectivity graph where nodes represent parts and edges indicate physical connectivity between them. We create this graph automatically based on the proximity of parts.

Designs often feature complex moving parts connected by mechanical joints. Such connections are used in various doors and drawers, and in complex moving objects like swings, wheels, and steering assemblies. We rely on the experts to annotate their designs with this functionality if it exists. We store moving components in the standard way: as a hierarchy of joint transforms along with their respective joint types (prismatic, ball joint, hinge joint) and joint limits.

Domain experts also annotate parts that are purely structural (e.g. screws, hinges, and brackets), henceforth called *connecting parts*. The *connecting parts* are separated from the *principal parts* of the design (e.g. shelves, legs, wheels) since the connecting parts will not be used explicitly in the design-by-example process. Instead, we relieve the user from the tedious task of specifying them by inferring and adding them automatically during the design process (see Section 5.6).

3.3 Parametric Manufacturable Designs

From the input collection of manufacturable models created by experts, we automatically generate a parametric representation and then augment this representation by incorporating information on how elements connect to each other in the physical space.

The parametrization we propose is inspired by two classes of methods. The first class preserves global relationships [Gal et al., 2009, Kraevoy et al., 2008, Zheng et al., 2011a], but only considers continuous variations in the shape. The second class allows discrete variations [Bokeloh et al., 2012a], but only accounts for local relationships. We combine these two ideas to construct models that both preserve global relationships such as symmetry *and* perform topological changes to preserve discrete regular patterns. Like Bokeloh et al. [2012a], we construct a linear model that allows expressing the space spanned by all possible manipulations as a parameterized model. The parametric models in this collection possess two additional qualities that are essential for data-driven design for manufacturing applications. First, they follow a hierarchical tree structure that allows assembly of new models by composing substructures at different levels of the tree. Second, they encode information that guarantees fabricability—for example, we represent how parts connect to each other in the physical world, and we allow parts to be resized only if a corresponding physical process is possible.

An example of a parametric model is depicted in Figure 1-3. The figure illustrates the large amount of geometric variety that a single parametric design can encode: in the figure, a cabinet becomes everything from a workbench to a nightstand.

More formally, a parametric design T^i at the i^{th} level of the hierarchy can be written as

$$T^i = \left\{\mathcal{F}^i, \mathcal{A}^i\right\} \tag{3.1}$$

where $\mathcal{A}^i$ is the feasible set that constrains the parameter values to ensure manufacturability, and $\mathcal{F}^i$ is a function from parameter values $\mathbf{q}^i \in \mathcal{A}^i$ to a geometry (e.g., a mesh).

Our method converts each design to a parametric tree, following the hierarchical representation determined by the experts. For each leaf node, we explicitly define $\mathbf{q}^i$ and $\mathcal{F}^i$, and we define $\mathcal{A}^i$ based on the set of constraints that act on $\mathbf{q}^i$. For the internal nodes, we specify $\mathbf{q}^i$ and $\mathcal{F}^i$ as the composition of the children nodes. The feasible set on an internal node can be defined as the intersection of the feasible sets of its children restricted by additional "coupling constraints" that bind multiple parametric designs together.

3.4 Automatic Hierarchical Parametrization

Our method for automatically converting a design from the collection to a parametric design comprises two steps. First, we select the leaf nodes and assign their degrees of freedom $\mathbf{q}^i$ and deformation functions F^i. Second, we analyze the geometric-semantic (*geosemantic* Shtof

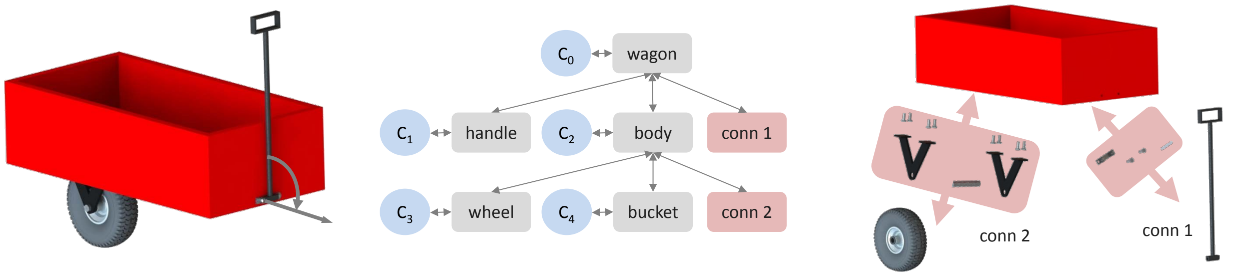


Figure 3-2: From left to right: a design example of a toy wagon, the hierarchical tree, and a visualization of the connections. The arrow on the handle indicates that this part has an articulation, namely that it can rotate along the depicted axis. The tree includes the geosemantic relationships that are stored at each level of the hierarchy, C_0 to C_4 (shown in blue), as well as connections (depicted in red). The visualization on the right illustrates the information contained in each connection node.

et al. [2013]) relationships between parts of our model in order to define structure-preserving constraints at each level of the hierarchy, thus determining $\mathcal{A}^i$ (see example in Figure 3-2).

We use the hierarchical structure of a design to guide the construction of the parametric tree. In our representation, a leaf-node in the tree serves as a "least-fabricable-unit", the simplest single entity that can be constructed. Leaf-nodes play a crucial role in the remainder of our algorithm, and are therefore referred to as *elements* to clearly distinguish them from the internal nodes of the hierarchy. Elements that correspond to principal parts are called *principal elements*, while elements that correspond to connecting parts are called *connecting elements*.

3.5 Defining the mapping function F

In most cases, each part P^i in the design is assigned to be a leaf-node. We then choose $\mathbf{q}^i$ to be the 6-vector composed of the 3-dimensional position of the center of the part and the three axis-aligned scaling parameters. The deformation function $\mathcal{F}^i$ simply applies the prescribed scale and translation to P^i.

We also allow leaf nodes to represent repeating patterns of parts, such as an array of screws or a row of wooden planks (see Figure 3-3). The algorithm selects regular patterns by searching for identical parts that have uniform inter-component spacing in one or two dimensions. Parts are said to be identical if they have the same corresponding item and the same dimensions. The search is performed by first grouping all identical parts and then extracting all the non-intersecting subsets of that group which form a regular pattern. The algorithm selects the subsets in order of number of parts (highest first) to guarantee that the non-intersecting rule does not prevent us from extracting the larger sets.

Following the notation of Bokeloh *et al.* [2012a], we consider only translational patterns. One-dimensional patterns are parameterized by $(\mathbf{o}, l, \mathbf{p})$, where $\mathbf{o}$ is the center of the first part (with respect to the bounding box of the element), l is the number of parts, and $\mathbf{p}$ is the generator translation. Two-dimensional rectangular patterns can be represented in the same manner.

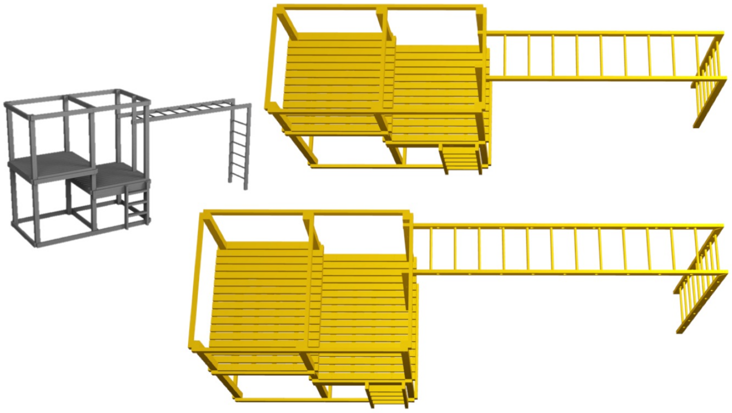


Figure 3-3: A parametric model with pattern elements. Upon resizing, both the number of floor planks and the number of rungs in the monkey bars change.

Given $\mathbf{q}_i$, the deformation function $\mathcal{F}_i$ calculates the new optimal values of l and $\mathbf{p}$, which we call $\bar{l}$ and $\bar{\mathbf{p}}$, respectively. We choose $\bar{\mathbf{p}}$ to be as close to $\mathbf{p}$ as possible. In the case of connecting elements, the algorithm is conservative when computing $\bar{l}$ in order to guarantee manufacturability. However, it ensures that $\bar{\mathbf{p}}$ does not shrink to the point that parts intersect. Finally, if the parts have resizable dimensions in the directions of $\mathbf{p}$, the are scales according to $\bar{\mathbf{p}}$.

3.5.1 Geosemantic Relationships

To define $\mathcal{A}^i$, we constrain the space of parameter variations by extracting geosemantic relationships from the design and ensuring that they are preserved when the parameters $\mathbf{q}^i$ are manipulated. Following the ideas described in Chen et al. [2013], Gal et al. [2009], Zheng et al. [2011a], we take into account the following types of relationships between elements: concentricity, coplanarity, and symmetry. In addition, we consider relationships in the order of the elements. Preserving order relationships guarantees that elements do not penetrate each other or exchange position when the parameters are modified.

Extraction of geosemantic relationships is guided by experts' annotations. First, the method takes into account the articulation information to ensure that all geosemantic relationships hold for all design configurations. Specifically, it considers the poses in which articulated parts reach their joint limits. Second, it takes into account the physical properties of elements in the design using information from the corresponding items in the items catalog. For example, it constrains the scaling parameters of elements that are linked to items that cannot be resized in a certain dimension.

The geosemantic relations are stored in the hierarchy at the lowest internal node that is a parent of all the related elements. This allows the use of any sub-tree in the hierarchy by itself since its defining relations are self-contained. Consider, for example, the toy wagon in Figure 3-2. The leaf-nodes store scaling constraints on each element (C_1, C_3, and C_4). The body assembly stores the coplanarity and concentricity relationships between the wheel and the bucket (C_2). Finally, the root node stores the coplanarity relationships between the handle and the bucket (C_0). Notice that the handle has an articulation; therefore, we compute relationships in both the horizontal and vertical rest configurations.

We characterize geosemantic relationships as functions that act on the bounding box or *prominent planes* of each element. In most cases, the six planes that define the bounding box to be the prominent planes are chosen. Since both the box and the planes are determined by $\mathbf{q}^i$, each geosemantic relationship can be expressed as a linear equality or inequality that constrains $\mathbf{q}$. Since the method stores the geosemantic relationships hierarchically, we can construct a linear system at each node T^i by aggregating the constraints of all its children. The feasible set $\mathcal{A}^i$ is then the set of all solutions to that linear system. Note that all of these relationships are computed automatically based on the geometry of the original design and the annotations on the input data.

For an illustration on how to extract geosemantic relationships and represent them as linear constraints, consider the table in Figure 3-4 (in 2D for simplification). It consists of three elements, namely, the tabletop and two legs. The parameters of each element are $\mathbf{q}^i = [p_x^i, p_y^i, \Delta_x^i, \Delta_y^i]$, which correspond to positions and sizes in each dimension. Rules for coplanarity will stipulate that the bottom of the tabletop must coincide with the top of the legs, i.e., $p_y^{Top} - \frac{\Delta_y^{Top}}{2} = p_y^{Leg1} + \frac{\Delta_y^{Leg1}}{2} = p_y^{Leg2} + \frac{\Delta_y^{Leg2}}{2}$. Rules for concentricity constrain the center of the bounding boxes in each dimension. In rules for order, we simply replace the equality by an inequality.

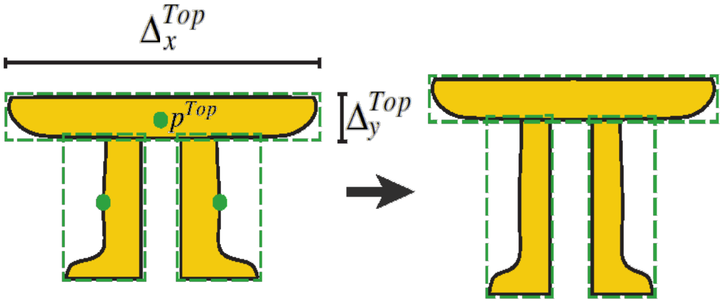


Figure 3-4: We show a simple 2D table consisting of three parts, a top and two legs (Leg1 and Leg2). Each part is contained within a single element for which the $\mathbf{q}^i$ are the positions (x, y) and sizes $(\Delta x, \Delta y)$ of the bounding box of the part.

Notice that the legs have a reflective symmetry. To write this relationship as a function of $\mathbf{q}$, we need to find a third element that has a center on the symmetry plane. In this case, we observe that this is true for the tabletop. Hence, if $\mathbf{n}$ is the normal of the symmetry plane (in the example, $n = [1\ 0]^T$), we can write down symmetry relationships as $\frac{(p^{Leg1}+p^{Leg2})^T\mathbf{n}}{2} = (p^{Top})^T\mathbf{n}$. (In the example, $p_x^{Leg1} - p_x^{Top} = p_x^{Top} - p_x^{Leg2}$.)

While concentricity and symmetry constraints are written directly as functions on the bounding box, coplanarity relationships are defined on the prominent planes of the model. In most cases, we use the six bounding planes as the prominent planes. However, in elements with more complex geometry, such as the go kart frames, we extract additional planes to describe relationships (for example, the plane on the axle that connects to the wheel). In this implementation we have manually annotated such prominent planes, but one could use simple geometry processing tools to infer them automatically.

We also create additional prominent planes in the case in which we have functional elements that assume multiple rest configurations. In this case, we create planes for every rest pose. We consider the centers of the joints $\mathbf{q}_c$. For every resting pose j, we can write the variables of the transformed element as linear combinations of $\mathbf{q}_i$ and $\mathbf{q}_c$. By adding $\mathbf{q}_c$ to the $\mathbf{q}$ vector, we can write $\mathbf{q}_i^j = \mathbf{V}\mathbf{q}$ and proceed to add constraints in the standard linear notation described above.

3.6 Connections

Fabrication requires not only tracking abstract geometric constraints, but also understanding where and how elements connect to each other in the physical world. To accomplish this, we augment the representation with nodes that keep track of the physical contact and connections between the principal elements. We represent these relationships as *connection* nodes. Connections include references to the data that will be manipulated and carried over when combining elements to compose a new model. These include:

- the set of principal elements that are in contact (usually two but sometimes more),
- the set of connecting elements that are responsible for holding the principal elements together,
- the set of geosemantic relationships between the connecting elements and the principal elements, and
- a set of "soft" geosemantic relationships for placement of connecting elements (discussed below).

Grouping the elements into such structures is straightforward and can be achieved by directly analyzing the connectivity graph of the design described in Section 3.2. The set of "soft" relationships encode additional constraints on the connecting elements. At each connection, we compute the contact of the principal elements (the *contact patch*), and add linear constraints to ensure that the dimensions and position of the connecting elements are preserved with respect to this contact patch. When the designer manipulates the models by changing the parameters of the principal elements, the parameters of the connecting elements are optimized using these additional relationships as soft constraints. This relieves the designer from the tedious task of manipulating each connecting element individually.

Like geosemantic relationships, connections are stored on the lowest internal node that is a parent of all the principal elements they reference. In the example of Figure 3-2, we have two connections. One connects the handle to the bucket and is stored in the root node of the toy wagon; the other connects the bucket to the wheel and is stored in the internal node that groups the body assembly.

Constructing the hierarchy in this manner ensures that each node in the hierarchy is a complete representation that depends only on its children. Therefore, the database includes not only full models of the original design (root nodes), but also all the other nodes in each hierarchy, representing parts and sub-structures. The result is a much richer database that supports the design-by-example mechanism of assembling new models by composition of parametric designs representing parts at various levels.

3.7 Discussion

With the help of domain experts, we have built what we believe to be the first open collection of manufacturable designs. It typically takes experts many hours to design a complete model using a commercial CAD software package. The simplest model in our database took approximately one hour to design, and the most complex (the go-kart) took three months. In this context, the time that is required to add the few annotations described in Section 3.2 (between 10 to 20 minutes) is almost negligible. More time is needed to create the items catalog: most of that time is spent finding suppliers for each of the items. But designers *always* need to find suppliers if they want to manufacture their models. Also, creating the items catalog can be viewed as a preprocessing step, because, once completed, the items catalog can be reused for subsequent designs.

We have proposed a representation that is parametric in addition to being hierarchical and fabrication-aware. By allowing only shape manipulations that correspond to parameter variations, we ensure that any model created by manipulating or mixing and matching component of this database can be directly manufacturable—we can automatically generate a bill of materials for fabrication. In the next chapter we will discuss the fundamental problem of retrieval in such parametric collection.

Another important aspect of this representation is that it handles the connecting elements differently from principal parts. While principal parts have degrees of freedom that have to be specified by the user, the parameters for the positioning and scaling of connecting parts are completely determined by the hard and soft constraints on the connections. We will use this representation to propose a new system for assembly-based design that ensures manufacturing and relieves the users from the tedious task of dealing with connecting parts (Chapter 5).

In this database, the parametrization was automatically generated for single CAD models from geometric-semantic relationships. An advantage of this approach is that it allows an explicit representation of the mapping function F from parameter values to a geometry—we can describe each shape as a 3D mesh where the vertices of each triangle can be expressed as a function of the shape parameters. This approach, however has two limitations. First, we use a linear representation of our templates, because this allows for solving for parameter constraints using simple linear and quadratic programs, which can be efficiently computed. Though this computational efficiency is essential in an interactive system, the linearity condition constrains our manipulations to scaling and translation of parts, since rotations are non-linear. Second, we base most of our computations on coplanarity relationships between prominent planes. This works well for many man-made models, but would present difficulties with models that have a more curved or complex geometry.

An alternative approach would be to use CAD parameters directly, which can represent more complex shape variations. CAD models are parametric from construction, allowing engineers to specify fabrication-aware constraints manually. The challenge in using such models in interactive tools is that the mapping function from parameter values to a geometry requires making an external call to CAD systems, which is computationally expensive. In Chapter 6 we describe a method that allows real-time evaluation of models that are parametrized directly using commercial CAD tools.

Chapter 4

Retrieval on Collections of Manufacturable Designs

4.1 Introduction

As previously discussed, data-driven design requires partial shape deformation for constraint satisfaction and part composition. In the context of manufacturing, these shape manipulations need to preserve structure and a variety of feasibility constraints. By explicitly defining a feasible set in parametric space, the parametric representation allows for a large variability while guaranteeing validity. This is especially important to support customization by users. Different users in different circumstances may require different designs of the same object, or may want to explore different variations of a similar design. In many cases it is impractical to explicitly design a new model for each variation, and this is where parametric designs are most useful.

A fundamental problem in data-driven design is the retrieval of shapes from a large collection. While shape-based matching and retrieval have been widely addressed for simple (non-parametric) shape databases, little progress has been made in efficient retrieval on collections of parametric shapes. In this chapter, we propose a strategy for searching through a database of parametric models in which the input query is expressed as a single 3D shape.

Retrieval on parametric shape collections is challenging because the search space is both discrete (number of shapes) and continuous (parameters values). In all previous work, when matching a given query model to a parametric model, one first has to fit the parameters to best match the query and then compute the distance from the query to the fitted shape. We call this the *fit-first* scheme. This scheme has several disadvantages. First, the process of fitting is time-consuming and has to be done for every shape in the database. It therefore does not scale well as the size of the database increases. Second, this scheme does not allow the use of descriptor space representations that have been shown to be effective for retrieval in a single (non-parametric) shape collection. The typical approach for efficient search does not rely on directly comparing a query element with every element in a database, but rather on pre-computing descriptors for each shape and then performing fast retrieval by computing distances in this high-dimensional descriptor space. Because the actual geometry of each parametric shape is known only after fitting, it is not possible to perform the time-

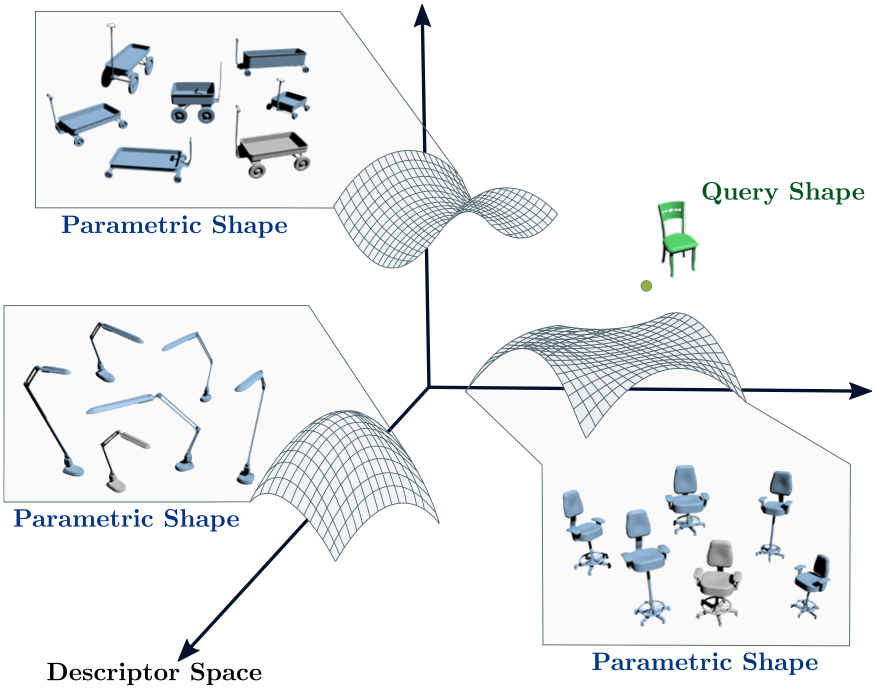


Figure 4-1: We propose a method for shape retrieval from parametric shape collections that uses a descriptor space representation. While shape descriptors map single shapes to points in a descriptor space, smooth descriptors map parametric shapes to low dimensional manifolds in this space. Our method efficiently represents these manifolds in order to allow for accurate and fast retrieval of the closest parametric model to a given query shape.

consuming task of computing descriptors a priori using the *fit-first* scheme. Descriptors must be extracted just before comparing or a direct comparison of the geometry must be used.

We propose a method for performing matching and retrieval from a collection of parametric shapes that does not follow the *fit-first* scheme. The key idea is to represent the full parametric shape, including continuous variations, in descriptor space. While single shapes can be described as points in a descriptor space, parametric shapes occupy larger "regions". To find the closest parametric shape given a query model (single point in descriptor space) we need to efficiently compute the distance from this point to each shape "region" and retrieve the closest one. We address this problem by creating a compact representation for these regions that allows minimal storage and fast evaluations, all the while guaranteeing accurate distance measurements. We observe that, for smooth descriptors, these regions are bounded low-dimensional manifolds embedded in high-dimensional space. The dimensionality of these manifolds is given by the number of parameters and the bounds are given by the feasible set of parameter values. We also have access to the actual function that defines the manifold, given by the composition of the parametric shape function and the signature function of the descriptor (see Figure 4-2).

We propose an algorithm for covering each manifold with a set of primitives that can be efficiently used for retrieval. We use two types of primitives: *points* and *bounded tangent spaces*. We discuss methods for creating these primitives (specifically, defining the bounds for the tangent spaces) and selecting between them to guarantee efficient storage and retrieval.

The general idea is that flatter regions should be covered by tangent spaces while more curvy ones should be covered by points. However, since different primitives have different storage and retrieval costs, the optimal cover depends not only on the geometry of the manifold, but also on the desired amount of accuracy. We therefore define an approximation error for our classification application and propose a method for primitive selection based on curvature, boundary evaluation, and allowed approximation error. The theoretical analysis allows me to compute threshold values with no need for empirical parameter adjustments and provides guarantees on retrieval accuracy.

4.2 Related Work

Shape Retrieval Efficient retrieval of 3D shapes has drawn the attention of the graphics community for many years. For a survey of shape retrieval methods we refer the reader to [Tangelder and Veltkamp, 2008]. For more recent advances on the field we refer the reader to [SHREC, 2014]. One of the most common approaches for fast retrieval is the use of descriptors that represent geometric models as points in a high dimensional feature space. In this approach the main computational cost is performed in preprocessing by evaluating the descriptors for each shape. Retrieval at run-time is reduced to a high dimensional nearest neighbor search in descriptor space that can be performed quite efficiently. There is a vast literature on descriptors for 3D shapes, ranging from simple histogram methods [Osada et al., 2001] to light transport functions [Chen et al., 2003]. Benchmarks for comparing these descriptors have also been proposed [Shilane et al., 2004] and the choice of descriptor is usually done based on the tradeoff between accuracy and computational cost. Some approaches also propose descriptors that are independent of certain shape transformations, such as articulations [Bronstein et al., 2011, Gal et al., 2007]. However, none of these methods can capture the variability of parametric shapes. Parametric models that return a different geometry for different parameter settings cannot be represented as points since they cover large regions of the descriptor space. We propose a method to efficiently represent these regions.

Template-driven Shape Exploration This problem is also related to works that represent a category of discrete designs using a parametric 3D template. In this case, the template is not a parametric design, but a description that generalizes a set of models. Ovsjanikov et al. [2011] construct a single template to generalize a particular shape category and use it to explore the variability of the collection. Kim et al. [2013] produce a set of probabilistic templates that group large shape collections into clusters that capture the shape variations. These exploration tools have also been used for shape synthesis [Averkiou et al., 2014]. Finally, Yang et al. [2011] propose a method for exploring meshes with similar connectivity while preserving constraints. Although these works do not directly address the retrieval problem, some of the proposed techniques relate to our problem. A key observation of Ovsjanikov et al. [2011] is that since templates have a low dimensional set of parameters, they lie near a low dimensional manifold in a descriptor space. Following this observation, they use PCA to extract the variability of the shape collection in this space and use optimization to convert it into the variability of the template deformation. Similarly, Yang et al. [2011], define the shape space as a manifold which is navigated by local planar and quadratic approximations. In line with these works, we represent our parametric shapes as low dimensional manifolds in descriptor space. However, in our approach each manifold is defined by a single parametric shape and not a set of non-parametric shapes. Moreover, we aim to represent a collection of such manifolds, defined by a collection of parametric shapes, and support distance queries from all of them to allow efficient retrieval.

Point Clouds in High Dimensions Since we represent parametric shapes as low dimensional manifolds in a descriptor space, this work is related to compact representations of low dimensional data in high dimensions. Manifold learning is a strategy that aims at finding meaningful low dimensional structures in high dimensional data using non-linear dimensionality reduction methods such as ISOMAP [Tenenbaum et al., 2000] and LLE [Roweis and Saul, 2000]. In these approaches, we assume that the K-dimensional manifold is represented as a point cloud in an N-dimensional space ($K \ll N$) and no additional information is known. The result of such techniques is a map $A : \mathbb{R}^N \mapsto \mathbb{R}^K$ that allows projecting points into this low dimensional space. This representation however cannot be used for retrieval since distances to query points must be computed in $\mathbb{R}^N$ allowing comparisons across manifolds.

By creating a point cloud representation of each parametric shape using sampling in parameter space, our problem is closely related to a classification problem in high dimensional data, where each parametric shape defines a class. Among the most common approaches for this problem are Gaussian mixture models [Bishop, 2006] which can be computed using EM algorithms. Since parametric shapes are low dimensional, Gaussians in $\mathbb{R}^N$ cannot compactly cover each shape space and additional dimensionality reduction would be necessary to guarantee minimal overlap between class representations. Alternatively, one can use a method such as mixtures of factor analyzers [Ghahramani et al., 1996], which concurrently performs clustering and local dimensionality reduction within each cluster. In this application, however, instead of starting with a point cloud, we have access to the actual function that defines the manifold, namely, the parametric model composed with the descriptor evaluation. We also know the underlying dimensionality which is defined by the number of parameters. We take advantage of this in our algorithm, measuring geometric properties such as derivatives and curvatures on sampled points, which are not present in a point cloud representation.

Distances to Manifolds Our approach relies on an estimate of distances from points to manifolds with a known parametrization map, a problem that has also been addressed in several research areas. Pottmann and Hofer [2003] propose a method for constructing smooth functions that approximate the distance from a a point $x \in \mathbb{R}^N$ (variable) to a given manifold (fixed). These functions have second order accuracy with respect to x and can therefore be used in optimization tools which have the position x as a variable and the distance to the manifold as part of the cost function. This has been applied, for example, in the context of registration [Mitra et al., 2004, Pottmann et al., 2004] and surface approximation [Wang et al., 2006]. In this work however, since in any retrieval experiment the x position is given by a query shape (fixed), second order accuracy with respect to x adds no information to our measurement. Instead we prefer simpler functions that approximate the first order distance metric and allow for fast estimation of the closest manifold to the query point. This involves efficient representation of the manifold to allow for fast distance estimation given a fixed query point. Vural and Fossard [2011] have proposed a method for discretizing manifolds to allow for distance estimation and classification. Their algorithm has similar goals: they sample each manifold, all the while attempting to determine the number of samples that should be retained to maximize classification accuracy. Their work, however, is restricted to point sampling. Tangent approximations have also been widely used to approximate manifold distances [Srivastava et al., 2005, Vasconcelos and Lippman, 2005]. These are known to provide a more compact representation, but only work locally since they are equivalent to the first order Taylor approximation. In our approach we combine these two ideas by proposing a hybrid approach where the manifold is represented by a set of primitives that can be either point samples or bounded tangent spaces. In our method we address the question of how to select between the primitive types in order to optimally allocate resources and discuss theoretical and empirical bounds on retrieval accuracy.

4.3 Representation of Parametric Shapes

As in Chaper 3, we define a parametric shape as $T = \{\mathcal{F}, \mathcal{A}\}$, where $\mathcal{A} \subset \mathbb{R}^K$ is the feasible set that constrains the parameter values, and $\mathcal{F}$ is a function from parameter values $q \in \mathcal{A}$ to a geometry (e.g., a mesh).

Given a query shape s, we would like to compute the distance from s to T. Formally, this distance is defined by:

$$\text{dist}(s, T) = \min_{q \in \mathcal{A}} \left(\text{dist}(s, \mathcal{F}(q))\right),$$

where the distance between two fixed shapes $\text{dist}(s, \mathcal{F}(q))$ can be defined by a given shape descriptor. However, instead of finding the optimal value of q and computing the distance for this parameter (i.e., *fit-first*), we will find this distance by defining a representation in a descriptor space of the whole parametric shape. Similar to the previous work Chen et al. [2003], Osada et al. [2001], we represent a geometry using a descriptor that takes a 3D mesh and computes a signature vector (typically signature vectors are high dimensional). This signature vector compactly represents a single geometry as a high-dimensional point in a descriptor space. However, this approach is not obviously applicable to parametric shapes because parametric shapes span a large set of possible geometries and therefore occupy a larger region of the descriptor space.

As shown in Figure 4-2, we define $\mathcal{M}(q) = (\mathcal{D} \circ \mathcal{F})(q)$, where $\mathcal{D}$ is the signature function that generates a descriptor for a given geometry. We can interpret $\mathcal{M}(q)$ as a parametrization from $\mathcal{A} \subset \mathbb{R}^K$ to $\mathbb{R}^N$, where the number of shape parameters K is much smaller than the dimensionality of the descriptor space N. Our method assumes that $\mathcal{F}$ is smooth. This holds for the models that are automatically converted from single geometries and for most CAD models since these shapes are typically designed such that parameter variations smoothly deform geometries. As a result, for smooth descriptors we can assume that the image $\mathcal{M}(\mathcal{A}) = \bigcup_{q \in \mathcal{A}} \mathcal{M}(q)$ lies on a manifold. Therefore, given a query shape s, we can apply the signature function to compute its value in descriptor space $x = \mathcal{D}(s)$ and define

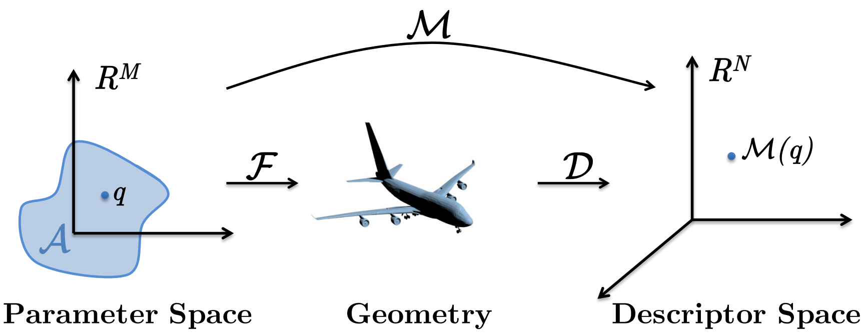


Figure 4-2: The function $\mathcal{M}(q) = (\mathcal{D} \circ \mathcal{F})(q)$ is a composition of the mapping function $\mathcal{F}$ from parameter values to a geometry with the signature function $\mathcal{D}$ which generates a descriptor for a given geometry.

$\mathrm{dist}(s, T) = \mathrm{d}_2(x, \mathcal{M}(\mathcal{A}))$, where d_2 is the euclidean distance in $\mathbb{R}^N$.

Our goal is to efficiently evaluate the distance from x to a collection of manifolds that represent each parametric shape in our database in order to retrieve the closest one (see Figure 4-1). Our approach is to construct a compact representation of each manifold that is an approximation with a certain allowed error. We aim at finding an approximation that has minimal storage requirements and allows for distance evaluations which are both fast and accurate.

4.3.1 Manifold Approximation

We approximate each manifold $\mathcal{M}(\mathcal{A})$ as a set of I primitives that locally describe the manifold: $\bar{\mathcal{M}}(\mathcal{A}) = \{P_1, ..., P_I\}$. For convenience, we will drop the parenthetical $(\mathcal{A})$ in the notation of $\mathcal{M}$ and $\bar{\mathcal{M}}$.

Our goal is to find the closest parametric shape in a collection given a query shape, i.e., find the closest manifold $\mathcal{M}$ given a query point x. Accordingly, we have a good approximation $\bar{\mathcal{M}}$ if the distance from x to $\mathcal{M}$ and the distance from x to $\bar{\mathcal{M}}$ are approximately the same. We therefore say the approximation error of the manifold is δ, if

$$\forall x \in \mathbb{R}^N, \quad |\mathrm{d}_2(x, \mathcal{M}) - \mathrm{d}_2(x, \bar{\mathcal{M}})| \leq \delta.$$

We can write this as:

$$\mathrm{d}_2(x, \mathcal{M}) - \delta \leq \mathrm{d}_2(x, \bar{\mathcal{M}}) \leq \mathrm{d}_2(x, \mathcal{M}) + \delta.$$

The inequality on the right is satisfied if:

$$\mathrm{d}_2(y, \bar{\mathcal{M}}) \leq \delta \ \ \forall y \in \mathcal{M}, \qquad \text{(Coverage Lemma)}$$

while the inequality on the left is satisfied if:

$$\mathrm{d}_2(\bar{y}, \mathcal{M}) \leq \delta \ \ \forall \bar{y} \in \bar{\mathcal{M}}. \qquad \text{(Tightness Lemma)}$$

of the Coverage Lemma. Given $x \in \mathbb{R}^N$, $\exists y \in \mathcal{M}$ such that $\mathrm{d}_2(x, \mathcal{M}) = \mathrm{d}_2(x, y)$. If the Coverage Lemma holds then there $\exists \bar{y} \in \bar{\mathcal{M}}$ such that $\mathrm{d}_2(y, \bar{y}) \leq \delta$. By the triangle inequality we get $\mathrm{d}_2(x, \bar{y}) \leq \mathrm{d}_2(x, y) + \mathrm{d}_2(y, \bar{y})$. Since, $\mathrm{d}_2(x, \bar{\mathcal{M}}) \leq \mathrm{d}_2(x, \bar{y})$ we conclude that $\mathrm{d}_2(x, \bar{\mathcal{M}}) \leq \mathrm{d}_2(x, y) + \mathrm{d}_2(y, \bar{y})$, which, in turn, gives us $\mathrm{d}_2(x, \bar{\mathcal{M}}) \leq \mathrm{d}_2(x, \mathcal{M}) + \delta$. □

of the Tightness Lemma . Analogous to the proof of the Coverage Lemma. □

The Coverage Lemma states that every point on $\mathcal{M}$ is sufficiently close to $\bar{\mathcal{M}}$. This means that every point on the original manifold can be represented by a point on our approximation. This guarantees that if $x \in \mathbb{R}^N$ is close to $\mathcal{M}$, then it will be close to $\bar{\mathcal{M}}$. The Tightness Lemma states that every point on $\bar{\mathcal{M}}$ is sufficiently close to $\mathcal{M}$, which means that there is no point on the approximation that is far from the manifold. This guarantees that if $x \in \mathbb{R}^N$ is far from $\mathcal{M}$, then it will be far from $\bar{\mathcal{M}}$. Together, the coverage and tightness mean that the Hausdorff distance between $\bar{\mathcal{M}}$ and $\mathcal{M}$ is bounded by δ.

4.4 Algorithm

Each primitive P_i of the approximation $\bar{\mathcal{M}} = \{P_1, ..., P_I\}$ is defined as either a *point* or a *bounded tangent space*, which is formed by the intersection of a tangent space at a given point with an ellipsoid $\mathcal{E}_i \subset \mathbb{R}^N$ centered at that point. We write:

$$P_i = \begin{cases} p_i \text{ or} \\ \{x \in \mathcal{E}_i | x = p_i + \sum_j a^i_j \mathbf{t}^i_j\}, \end{cases} \tag{4.1}$$

where p_i is a point on $\mathcal{M}$, $\{\mathbf{t}^i_1, \ldots, \mathbf{t}^i_K\}$ are the normalized directional derivatives which form a basis to the tangent space of $\mathcal{M}$ at p_i, and $a^i_j \in \mathbb{R}$ are weights.

To define $\bar{\mathcal{M}}$, we propose an algorithm that samples points y on $\mathcal{M}$ at random and then adds a primitive to $\bar{\mathcal{M}}$ if $D(y, \bar{\mathcal{M}}) > \delta$. Random sampling of points on $\mathcal{M}$ is done by randomly selecting points $q \in \mathcal{A}$ and computing $y = \mathcal{M}(q)$. The added primitive could either be a single point or a bounded tangent space as defined in Eq. 4.1. We argue that in the limit, this sampling algorithm assures that we get a complete coverage of the manifold. In our experiments we terminate sampling after 2000 rejections. This does not provide a technical guarantee of complete coverage, but it is a good approximation as shown in Figure 4-8. This is because the rejection sampling scheme we use will keep all points that are not covered by the approximation. Tightness is always satisfied when P_i is a point, but not when P_i is a tangent space. In this case, we use the Tightness Lemma to define a rule on how to determine the bounding ellipsoid $\mathcal{E}_i$ for P_i, as will be discussed below.

When our rejection sampling scheme chooses to add a primitive to $\bar{\mathcal{M}}$, the primary decision is to determine whether it should be represented as a single point primitive or a bounded tangent space with the point as its center. This choice is done to maximize efficiency. A bounded tangent primitive requires storing K tangent vectors in addition to the center point; we therefore say that its cost is $K+1$ times the cost of the point primitives. This also roughly corresponds to the increase in query computation time (see Section 4.5). Hence, if the bounds of a tangent primitive are tight enough such that the region it covers is smaller than the region that can be covered by $K + 1$ points, then it is not worthwhile to use this primitive. To make this decision we need to define and measure the coverage of both point and tangent primitives. We will first consider the case where the manifold is unbounded (i.e., $\mathcal{A} \equiv \mathbb{R}^K$) and later we will take into account the additional bounds imposed by the feasible set $\mathcal{A}$.

4.4.1 Unbounded Manifolds

If a manifold does not have bounds, the only aspect that determines how well it can be locally represented by tangent spaces is how much it deviates from being flat. We will measure how well a tangent space can locally approximate a manifold based on the Coverage Lemma. Then, we will discuss how we define the bounding hyperellipse $\mathcal{E}_i$ based on the Tightness Lemma.

Coverage First, let us consider the 1-dimensional case where $\mathcal{M} = c(t)$ is a curve in $\mathbb{R}^2$ and assume without loss of generality that the sample point is $p = c(0)$. In this case, the tangent approximation is then given by the line:

$$l(t) = c(0) + tc'(0),$$

Since we allow an error of size δ, then once a point is sampled, any point on the circle of radius δ centered at that point is well represented by the sampled point according to the Coverage Lemma. On the other hand, if we take the tangent line on that point, then any point on the curve that is within distance δ of this line is covered by the line representation. So while the coverage of a point is proportional to δ, the coverage of the tangent line is proportional to d, where d is the distance from the point p to the furthest point on the curve $c(t)$ that is sufficiently close to the tangent line (see Figure 4-3).

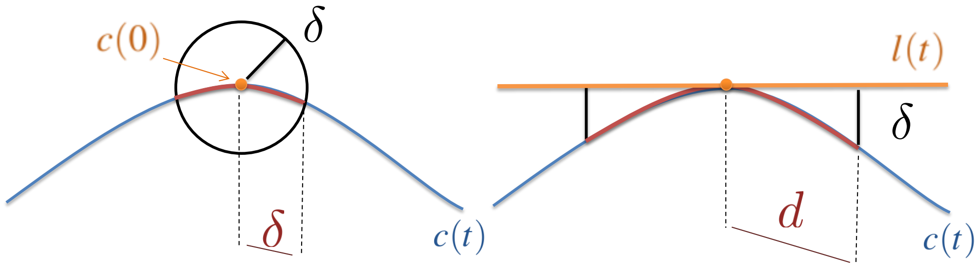


Figure 4-3: The coverage of a point (left) and of a tangent line (right) are defined by the region of the manifold (here illustrated as a curve $c(t)$) that is well approximated by this primitive given the allowed approximation error δ. While the coverage of the point $c(0)$ is directly proportional to δ, the coverage of the tangent line $l(t)$ is proportional to d, which depends on the curvature.

We can approximate d using the Taylor expansion. If $c(t) = c(0) + tc'(0) + \frac{1}{2}t^2 c''(0)$, then the distance from a point $c(t)$ to the line l is given by:

$$D(c(t), l) = \frac{1}{2}t^2 \left\| c''(0) - c'(0)\frac{\langle c''(0), c'(0)\rangle}{\|c'(0)\|^2} \right\|.$$

We can make this distance smaller than δ by bounding t as follows:

$$t \leq \sqrt{2\delta / \left\| c''(0) - c'(0) \frac{\langle c''(0), c'(0) \rangle}{\|c'(0)\|^2} \right\|}.$$

The distance d can then be approximated by $T_{\max}\|c'(0)\|$ from which we get:

$$d = \sqrt{2\delta / \frac{\|c''(0)\langle c'(0), c'(0)\rangle - c'(0)\langle c''(0), c'(0)\rangle\|}{\|c'(0)\|^4}}. \tag{4.2}$$

We observe that the denominator inside the square root of this expression is precisely the definition of curvature κ for the curve $c(t)$ at $t = 0$ Do Carmo [1976]. From this we can write $d = \sqrt{\frac{2\delta}{\kappa}}$.

A lower bound on the number of points needed to cover the same region as the tangent line is given by the ratio of the two coverages, d/δ. Hence, we should store a tangent primitive if this ratio is larger than the extra storage requirement, $K + 1$. That is, we should store a tangent primitive if:

$$k \leq \frac{2}{\delta(K+1)^2}. \tag{4.3}$$

This result is quite intuitive, since the curvature measures the amount by which the curve deviates from being flat. In our algorithm, we therefore measure the curvature at the point and if the curvature is small then we store the bounded tangent primitive, if it is too big we store the point primitive. The equation above defines how we determine this threshold based our the original parameter δ and the dimension of the parametric shape, so no additional empirical parameter estimation is needed.

This curvature interpretation can be easily expanded to $\mathcal{M} : \mathbb{R}^K \to \mathbb{R}^N$. In the multi-dimensional case, we use the maximal principal curvature Do Carmo [1976], which measures the curvature at the direction where it is maximized. Since the coverage ratio is now given by $(d/\delta)^K$, we get:

$$\frac{1}{k^{max}} \geq \frac{\delta}{2}(K+1)^{2/K}. \tag{4.4}$$

In our implementation, we approximate the maximal principal curvature k^{max} by the largest curvature in the K derivative directions. The curvature in each direction is computed using the expression for κ inside Equation 4.2, replacing the derivatives of curve c with the partial derivatives of the manifold $\mathcal{M}$.

Tightness To bound the tangent space we have to ensure that the Tightness Lemma is satisfied. As we did in the previous section, we will first look at the 1-dimensional case and will use the Taylor approximation. Then, the distance from a point $l(t)$ to the curve c can be bounded by the distance from a point $l(t)$ to the point $c(t)$ (see Figure 4-4a):

$$D(l(t), c) \leq D(l(t), c(t)) = \frac{1}{2}t^2\|c''(0)\|.$$

To ensure that this is smaller than δ we bound t, such that:

$$t \leq \sqrt{2\delta / \|c''(0)\|},$$

from which we get that the tangent space should be bounded by a circle of radius:

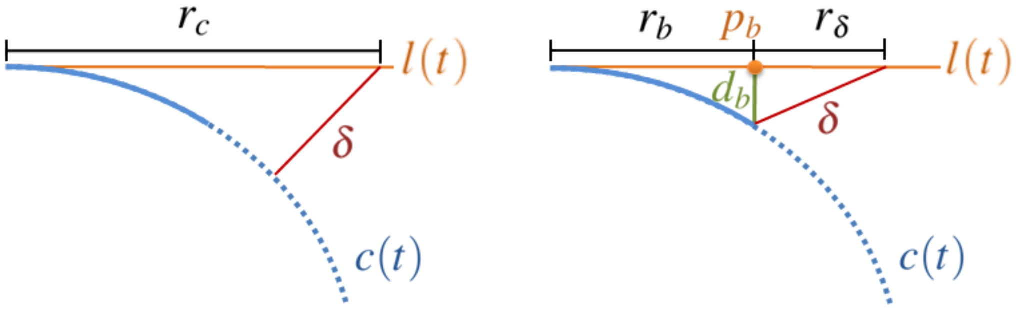
$$r_c = \sqrt{2\delta \|c'(0)\|^2 / \|c''(0)\|}.$$

Again, in the multidimensional case, we use a hypersphere and take the first and second derivatives in the direction of the maximal principal curvature.

4.4.2 Bounded Manifolds

Next, we discuss how to incorporate the feasible set $\mathcal{A}$ into our representation. Because the feasible set induces boundaries on the manifold in descriptor space $\mathbb{R}^N$, we need to incorporate this effect into $\mathcal{E}_i$ in order to guarantee tightness. This, in turn, affects the coverage of the tangent primitives and should also be taken into account when choosing which primitive to store.

Tightness Constraints Once again, we will start by looking at the one-dimensional case. As previously discussed, the curvature of the manifold defines a bound r_c to the tangent primitive, as shown in Figure 4-4a. We define boundary constraint r_b, as the largest radius that guarantees that the projection of the bounded tangent line onto the curve falls on points $c(t)$, such that $t \in \mathcal{A}$ (see Figure 4-4b).



(a) Unbounded case (b) Bounded case

Figure 4-4: Computation of the bounding radius for a tangent space primitive $l(t)$ on the manifold $c(t)$. In the illustration, the dotted line represents the part outside the boundary of the manifold and δ is the allowed approximation error. Left: when we do not take the boundary into account the bounding radius is determined uniquely by the curvature constraint r_c. Right: when we are close to the boundary, the radius is computed as $r_b + r_\delta$, where r_b is the distance to the boundary and r_δ is the amount by which we can expand the radius preserving tightness constraints. We can compute r_δ from δ and d_b, which is the distance from the boundary point p_b on $l(t)$ to the manifold.

If the point $y = c(0)$ is close to the boundary, then r_c could be larger than r_b. To guarantee tightness in this case, the tangent line has to be bounded by $r_b + r_\delta$, where r_δ is the amount by which we can expand the curve to guarantee that the distance from it to the bounded manifold is smaller than the allowed approximation error δ.

In the multi-dimensional case, we consider a direction $\mathbf{v}$ in descriptor space in which to compute the distance from a sample point $y = \mathcal{M}(q)$ to the boundary. We assume that we have a set of analytic expressions that represent the boundary constraints in the parameter space and then map them to the descriptor space using the Jacobian $\mathbf{J_q}$ at the sampled point $\mathbf{q}$. Then, if a boundary constraint in the parameter space is written as a function $g(\mathbf{x})$, in the descriptor space it becomes $\mathbf{J_q}g(\mathbf{x}) + \mathcal{M}(\mathbf{q})$. We can find the distance to the boundary $g(\mathbf{x})$ in the direction $\mathbf{v}$ by solving:

$$\min_{\alpha,\mathbf{x}} \|\mathbf{J_q}g(\mathbf{x}) - \alpha\mathbf{v}\|. \tag{4.5}$$

If the ray along the direction $\mathbf{v}$ intersects the boundary constraint $g(\mathbf{x})$, then the value of this minimization will be zero and the resulting α will return the distance from $\mathcal{M}(\mathbf{q})$ to this boundary constraint. The distance to the boundary, r_b, along $\mathbf{v}$ is then determined by computing this for every constraint $g(\mathbf{x})$ and taking the minimum.

To compute r_δ we first need to evaluate the distance d_b from the point $p_b = \mathcal{M}(\mathbf{q}) + r_b\mathbf{v}$ to the manifold. Then, as illustrated in Figure 4-4b, we can compute r_δ so that $\delta^2 = d_b^2 + r_\delta^2$. To compute d_b we use the second order Taylor approximation in a similar manner as explained above.

The computed distance to the boundary depends on the direction $\mathbf{v}$. Shooting rays in multiple directions and taking the minimum radius would determine a bounding hypersphere. This, however, is very restrictive, since a point can be close to the boundary in one direction and not in others. Therefore, we have chosen to bound the tangent spaces using ellipsoids instead of hyperspheres.

Naturally, the area covered by the ellipsoid depends on its orientation. Choosing optimal orientations for the ellipsoids can reduce the number of primitives needed to represent the manifold (see Figure 4-5). To determine a good basis for the orientation of the ellipsoids, we aim at aligning it with the least constrained directions of the manifold. We do this using a greedy approach. We start with a set of directions on the tangent space. First, we compute the distance from each of them to the boundary (using the method described above). Second, we take the direction that has the minimum distance to the boundary and set it as a basis vector. We then restrict our search to the orthogonal space of the current basis and repeat the first step. The algorithm ends after we complete a full basis.

Coverage To choose between points and tangent primitives, we first verify Equation 4.4 and then compare coverage taking into account the constraints imposed by the boundary. For each direction we set the coverage radius to be $r^i = \min(r_b^i + r_\delta^i, r_c^i)$. Then, following Equation 4.3, we choose to add a bounded tangent instead of a point if:

$$\prod_{i=1}^{K} \frac{r^i}{\delta} \geq (K+1). \tag{4.6}$$

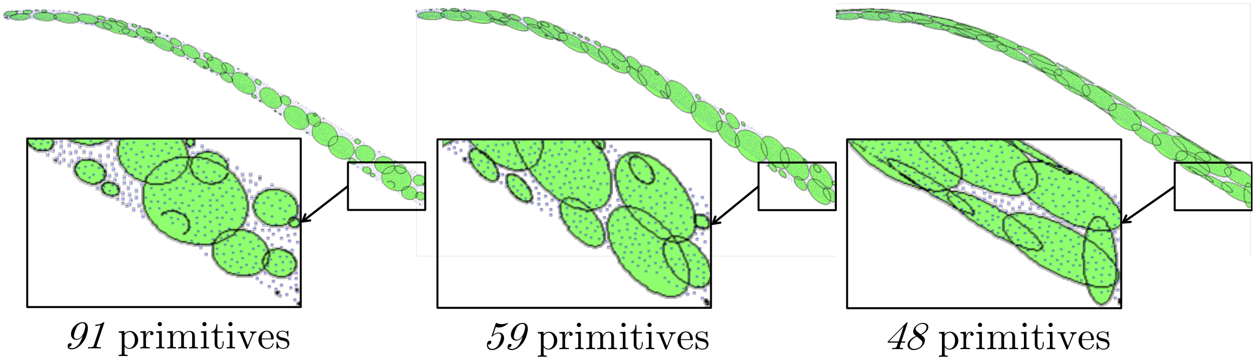


Figure 4-5: From left to right: covering the manifold with tangent spaces bounded by hyperspheres, non-oriented ellipsoids, and oriented ellipsoids. This example illustrates that the number of primitives needed to represent the manifold for the same value of δ is reduced when we use better primitives. We notice that even in this example with a 2-dimensional parameter space there is a significant improvement when oriented primitives are used. The blue dots represent the underlying manifold represented via super sampling. (Please note that these are high dimensional primitives projected to 2D for visualization and therefore appear slightly distorted.)

4.5 Retrieval

Our retrieval method determines the closest parametric shape by finding the closest primitive to the query shape. Distances to points are measured using standard euclidean norms and distances to bounded planes are measured by first projecting the query point x onto the tangent space and then computing the distance from the projection p to the ellipsoid. This distance is approximated using a scaling function S that maps the ellipsoid to the unit hypersphere centered at the origin. If $S(p) < 1$ then the distance is given by the projection error $d_p = \|x - p\|$. Otherwise, we approximate the distance from p to the ellipsoid as $d_e \approx \left\| p - S^{-1}\left(\frac{S(p)}{\|S(p)\|}\right)\right\|$ and thus the final error is given by $\sqrt{d_p^2 + d_e^2}$ (see Figure 4-6).

While computing the distance from a query point to a point primitive in R^N is $\Theta(N)$, computing the distance to a tangent space primitive involves additional computation for evaluating the projection onto the tangent space p and its distance from the ellipsoid d_e. We can precompute the NxK projection matrix for each tangent primitive and store it as part of our data structure. This does not affect our previous storage discussion since the ratio of the storage cost for tangent primitives as compared to points remains on the order of K. With this, computing the projection of the query point onto the tangent space is $\Theta(KN)$. Using the simplification discussed above, the computation of d_e is $\Theta(K)$. Therefore, while the distance to a point primitive is $\Theta(N)$, the distance to a tangent primitive is $\Theta(KN)$. From this, we conclude that, similar to storage requirement, the additional retrieval time for tangent primitives when compared to points is proportional to the number of parameters, K.

Though our method finds the closest parametric model to the query, finding the closest match still involves the final step of fitting parameters. Since we find the closest primitive, we

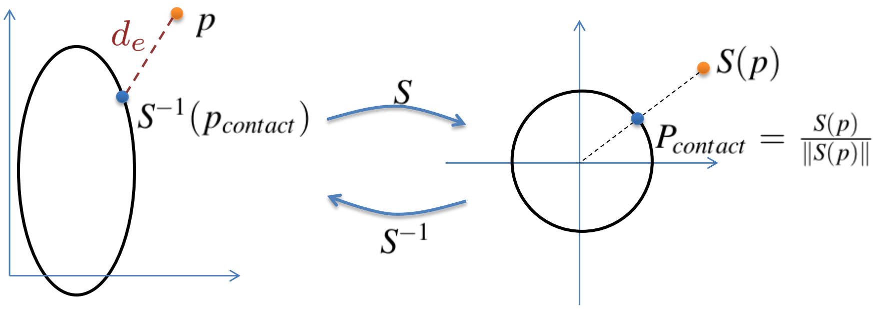


Figure 4-6: Approximating the distance d_e from the projected point p to the hyperellipse. Let S be a scaling function that maps the hyperellipse to the unit hypersphere centered at the origin. The point on the hypersphere that is the closest to $S(p)$ is given by $p_{contact} = \frac{S(p)}{\|S(p)\|}$. We use the inverse mapping and approximate the distance from p to the hyperellipse as $d_e \approx \|p - S^{-1}(p_{contact})\|$.

can use the parameters of this primitive as an initial guess and use existing search methods to refine it. This problem has been addressed in previous work with an iterative closest point (ICP) method Nan et al. [2012].

4.6 Experimental Setup

We tested my algorithm on a collection of models from multiple categories using three different descriptors.

4.6.1 Database

To run experiments with our retrieval method, we created a collection of parametric shapes using two procedures. First, we used a free CAD software (OpenSCAD) to model objects and expose designs parameters. Second, we used an automatic method to convert single geometries into parameterized objects (see Chapter refchap:3).

This collection of parametric shapes spans multiple categories, as shown in Table 4.1. We use 2 CAD designs and 74 automatically converted models. Figure 4-1 illustrates some of the models in the collection and their variations. The number of parameters for each design ranges from 2 to 9. We argue that this is a descriptive range, even considering complex parametric CAD designs. Although parametric CAD software allows for many independent variables, these are often constrained by manufacturing considerations and the need to interface with other models. Therefore, in practice, most CAD designs only have a small set of meaningful parameters that can be directly exported by designers to allow for further user-driven customization (typically less than ten).

For the parametric CAD models, the ranges for the exposed parameters were hand annotated by the designer. While generating the manifold representations in descriptor space, we call the CAD software to compute a 3D mesh for each parameter configuration. For models defined by our automatic conversion procedure, we represented each vertex of the mesh explicitly as a linear function of the parameters. This makes geometry evaluations very

Table 4.1: Parametric designs in our collection.

Category	Number of Models
lamps	17
boats	11
chairs	15
planes	9
carts	10
tables	15

fast, especially when compared to the CAD models where each evaluation involves several non-trivial operations.

In our designs, the feasible set of parameter values are linear: we defined ranges for exposed parameters of CAD designs and our automatic method defines the boundary of the feasible set using a set of linear constraints. With this assumption, the optimization shown in Equation 4.5 is a least squares problem that can be solved efficiently. We stress, however, that the mathematical model discussed in this paper does not depend on the linearity assumption. In addition, the implementation speedups given by the linearity assumption are only relevant during the pre-computation step that generates the manifold approximations and they do not affect retrieval time.

4.6.2 Descriptors

The algorithm we propose is independent of the choice of descriptor. The only assumption we make is that a descriptor should be quite smooth, so that the image of the parametric space lies close to a manifold in the descriptor space. We use three different descriptors for our experiments. The first one is the **D2 Shape Distribution**, which is defined by a histogram of distances between pairs of points on the surface of the model Osada et al. [2001]. The second is the **VOXEL Shape Histogram**, which is a shape histogram descriptor Ankerst et al. [1999] and describes the distribution of a model area as a function of the distances from voxel centers. Since these descriptors are not necessarily smooth, we interpolate the feature signal with a Gaussian kernel, following the approach of Ovsjanikov et al. [2011]. The third descriptor is the **Light Field Descriptor** Chen et al. [2003], which captures geometry detail from rendered images of the shape and is known to have good retrieval precision Shilane et al. [2004].

For the D2 descriptor, we sample $3,000$ points on the surface of the model and express them as a function of the shape parameters. For a given parameter setting, we measure the pair-wise distances (normalized by the average distance) between all sampled points and convolve this distribution with a set of Gaussian kernels of a fixed width σ and means distributed uniformly between 0 and 3. Since in our collection each sampled point can be written as a linear expression of the parameters, derivatives can be computed analytically. We have also experimented with finite differences, which are faster to compute and comparable in terms of accuracy. In our experiments we set $\sigma = 0.1$ and use 300 Gaussian means. We have also used PCA on our dataset to reduce the descriptor space to 24 dimensions.

For the VOXEL descriptor, we sample one million points on the surface for the model. We take the difference from each sample point to the center of mass and normalize them by maximal distance. We convolve this distribution with a set of isotropic 3D-Gaussian kernels that have variance σ and means distributed uniformly on a 3D grid. For this experiment, we made sure to resample all the points for each parameter configuration and use finite differences to compute first and second derivatives. We have used $\sigma = 0.2$ and 1000 Gaussians distributed on a 10x10x10 grid. As with the D2 descriptor, we use PCA on the dataset to reduce the descriptor to 64 dimensions. Though this descriptor is not rotation invariant, we exploit the fact that our models are CAD designs that have upright orientation and are aligned with one of the four principal axes. Therefore, we perform retrieval on four rotated versions of the query and keep the best one.

Construction of the Light Field descriptor involves transforming a model to be centered at the origin and inside of a unit sphere. The model is then rendered from a number of viewpoints, sampled from the vertices of a dodecahedron. Image features are computed as in Chen et al. [2003], combining Zernike moments with Fourier coefficients. Again, in this setting, we handle rotation invariance by performing retrieval on four rotated versions of the query and keeping the best one. We use PCA on the dataset to reduce the descriptor to 280 dimensions.

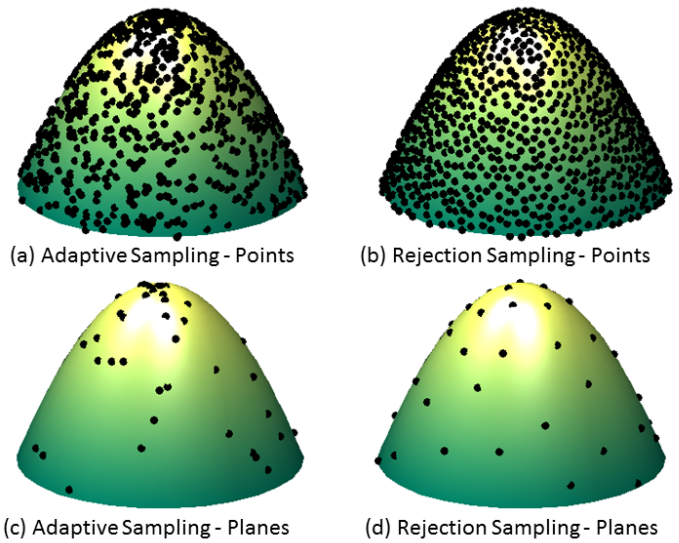
4.7 Evaluation

We present results on evaluating the accuracy and efficiency of our manifold representation for individual parametric shapes, as well as on the overall retrieval method from a shape collection.

4.7.1 Manifold Representation

Sampling Scheme Alternatives to our rejection sampling approach are adaptive sampling schemes based either on curvature or on surface areas. While a method based on curvature is suitable for approximating the manifold with tangent planes, adaptive sampling based on surface area approximates a uniform distribution of points in descriptor space. We have implemented both of these approaches using a Metropolis–Hastings algorithm, where the probability density function given a current sample is given by a Gaussian centered at that point with variance proportional to the curvature or surface area measured at that point. We compare the results from these approaches to our rejection sampling scheme using only point or tangent plane primitives and illustrate the results in Figure 4-7. We observe that we get similar distributions for curvature based adaptive sampling and rejection sampling for tangent primitives, and similar distributions for surface area adaptive sampling and rejection sampling using only point primitives.

However, while the adaptive sampling approaches are good at approximating the desired distributions, the randomness in the algorithm makes it unsuitable for minimal coverage, especially when the number of primitives is small. Figure 4-7 shows how points tend to clump together and leave gaps. On the other hand, the rejection sampling scheme guarantees that only points that contribute to coverage are added. In addition, our method determines the


(a) Adaptive Sampling - Points (b) Rejection Sampling - Points

(c) Adaptive Sampling - Planes (d) Rejection Sampling - Planes

Figure 4-7: Comparison between adaptive sampling and rejection sampling on a simple paraboloid example. Our rejection sampling scheme was done for both point and planes for a fixed approximation error δ. The number of samples for the adaptive sampling schemes was chosen to be the same as the result of the rejection sampling for both points and planes. The top row shows the results for point samples. Though both methods return a uniform distribution, in the adaptive sampling scheme points tend to clump together and leave gaps. The bottom row results for approximating with tangent spaces (we only display the center of the tangent space for simplification). Once again both methods display the desired distribution (based on curvature) and rejection sampling covers the space more effectively.

number of samples based on a unique user-specified parameter that reflects the retrieval error. Although the parameters of the adapting sampling schemes may be tweaked to vary how densely the sampling covers the space, these cannot be easily mapped to a global approximation error. Since sampling is part of a pre-processing step, this justifies a more expensive approach (rejection sampling) that results in lower storage, more controlled approximation error and faster run-time. Another important aspect is that the criteria for adaptive sampling depends on the primitive type while the rejection sampling method we propose can handle a hybrid representation. Finally, we can incorporate the boundary information to the rejection sampling algorithm, which allows me to create a compact representation for the bounded manifolds.

Accuracy We evaluate the accuracy of our manifold representation by measuring the actual *fitting* and *coverage* error for different values of the target parameter δ. We measure coverage error by sampling points from the ground truth manifold $\mathcal{M}$ and computing the distances to the representation $\bar{\mathcal{M}}$. We measure fitting error by sampling points on $\bar{\mathcal{M}}$ and computing distances to the ground truth $\mathcal{M}$. As ground truth we use a dense super sampling of the manifold $\mathcal{M}$, namely, a point-only (no tangent approximations) rejection sampling with very small $\delta = 0.005$. Figure 4-8 shows results of an experiment on a para-

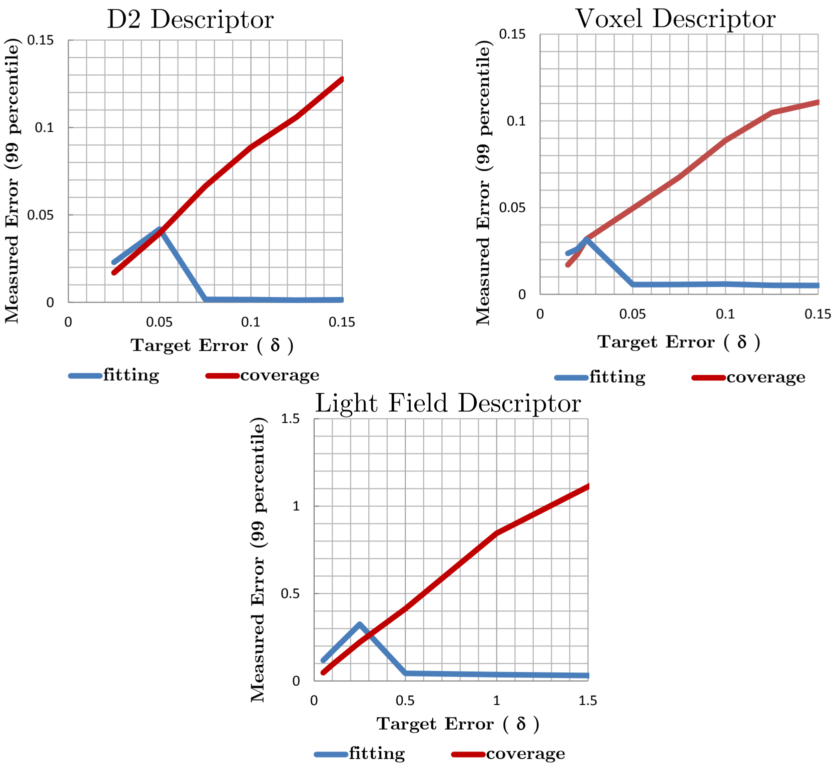


Figure 4-8: Measuring *fitting* and *coverage* errors as a function of the target parameter δ for the implemented descriptors. we observe that both measured errors are within the bounds of δ. For large values of δ we observe that the fitting error drops to zero. This is because for very coarse approximations, our algorithm prefers points to tangents – the coverage of points become larger with δ, while plane coverage is still limited by the curvature and the boundary of the manifold. Since absolute distance values depend on descriptors (and are much larger for the Light Field descriptor), the ranges of the target errors for this experiment were chosen so that the number of samples were similar for all descriptors.

metric chair with 2 parameters for all three descriptors. We plot across different values of δ the 99 percentile error (the worst error discarding the worst 1%).

Efficiency To evaluate the efficiency of our representation we compare our method with a rejection sampling scheme that uses just point primitives and one that uses just tangent primitives. Figure 4-11 shows the storage cost of each representation across different values of the target parameter δ (that defines the accuracy of the approximation). The cost depends on the number of stored primitives and their storage costs. We set the cost for point primitives to 1 and the cost of tangent primitives to $K+1$, where K is the number of parameters (DOF) of the shape. Note that counting ellipsoids as $K+1$ times more expensive also roughly corresponds to the increase in query computation time. We present evaluation results on three different shapes: a table with 3 DOF, a cart with 5 DOF and a chair with 7 DOF.

The top row in Figure 4-11 shows the cost verses δ on a log scale. We observe the trend of preferring tangents for small values of δ and points for large values of δ. As shown in the graphs, our hybrid representation can optimally select the transition between points and

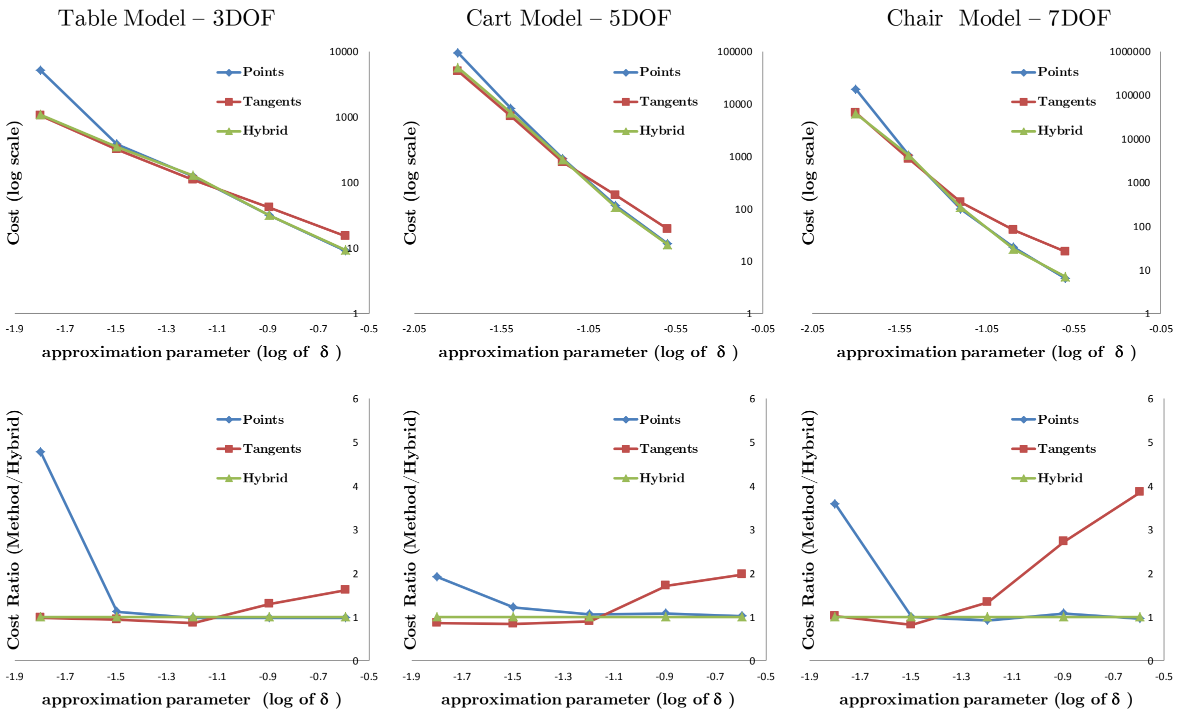


Figure 4-9: Comparison between our hybrid method and using a single primitive. Top row: shows the storage cost of each representation across different target parameters δ in log scale. Bottom row: the relative cost of the single primitive methods while compared to our method.

planes so that its cost is constantly bellow the other two alternatives. The second row shows the relative cost of using a single primitive compared to our method. As shown in the graphs, the relative cost of our hybrid method is close to one of the two at a certain δ range, while the other methods have a cost up to 5 times larger, depending on the shape and primitive type. The small oscillations in this graph are mostly due to the randomness in the sampling algorithm, but are also related to some approximations in our implementation.

Note that our representation uses both primitives and does not change between them at a specific point. It has a higher percentage of points for larger values of δ, and a higher percentage of ellipsoids for smaller values of δ. Note also that the cross-over value where point primitives start to outperform tangents varies depending on the shape. This happens because different shapes have different numbers of parameters and also because the sizes of the feasible sets vary. These parameters influence the coverage of a tangent approximation. Therefore, using a representation that picks one primitive type depending on the target parameter δ is not feasible as the transition value depends on the shape. Given a specific δ, some shapes may be better represented with points while others with planes. In contrast, our hybrid representation can adapt to the specific shape and choice of δ, automatically choosing the right combination of the two primitives. This is especially important while representing a collection of shapes, as only a hybrid scheme can optimize across all the different models in the database.

4.7.2 Retrieval

Next, we evaluate how well our representation works for retrieving models in a collection of parametrized shapes. First, we motivate the importance of taking into account the parametrized model. We evaluate what happens when we do not represent the full manifold but instead, use a shape with the default template parameters (we will call this the mean shape). For the query, we use random parameter configurations and search the database for the closest mean shape. We have used the D2 descriptor for this experiment. Figure 4-10 illustrates a few of the retrieved results. In this figure we show the query shape, the mean shape of the models that originates the query and the closest mean shape retrieved. We observe that the changes on the parameters significantly affect the geometry. For example, when we flatten a cart it resembles a coffee table; if we shrink the feet of a stool it resembles a lamp. We ran the mean shape retrieval test on 20 random parameter samples for each model in the database. In this experiment, we managed to retrieve the correct mean shape only 29% of the time.

We also compare our sampling scheme with the naive approach that represents the manifold by randomly sampling a fixed number of points from the *parameter space* of each model. The advantages of our approach are threefold. First, distributing samples uniformly over the *descriptor space* rather than over the parameter space provides better coverage of the manifold. Second, fixing the value of δ for each shape allows the number of samples per shape to vary according to the size of the corresponding manifold in the descriptor space. Third, by storing both points and tangent primitives we reduce the storage cost.

To compare the two methods we evaluate their performance on retrieval of points sampled from our parametric shape collection. Naturally, if we sample these points randomly over parameter space the naive approach will have a better performance on average since it matches the distribution. Alternatively, if we sample uniformly over descriptor space, our approach does much better on average. For a fair comparison, we sample uniformly over the parametric space, but evaluate the worst-case, rather than the average error, measured as the distance to the closest primitive on the correct manifold minus the distance to the closest primitive on a wrong manifold. As a result, the error does not depend on the query distribution, but on how well our samples cover the space. Since the error is measured in descriptor space and needs a reference to interpret, we plot it against the target error δ (also in descriptor space) for our method in Figure 4-11. For an "equal cost" comparison, we plot the error of the naive method using the same amount of storage as our method. We plot the results sampling both over the full database and over individual categories.

The results show that the error for our method is not only smaller (close to δ), but is also consistent. For example, airplane models present very small variations in the D2 descriptor space and can therefore be well represented with only a few samples. On the other hand, chair and lamp variations are much more dramatic, and therefore need more samples. Our method automatically handles this difference and allocates more storage to represent larger manifolds. We therefore notice that our method performs consistently better across all categories.

Finally, we show retrieved results for models in each category that were collected from online shape repositories. In these experiments, retrieval times for each model were in the order of 10 milliseconds. Figure 4-12 shows the top result for both descriptors for varying

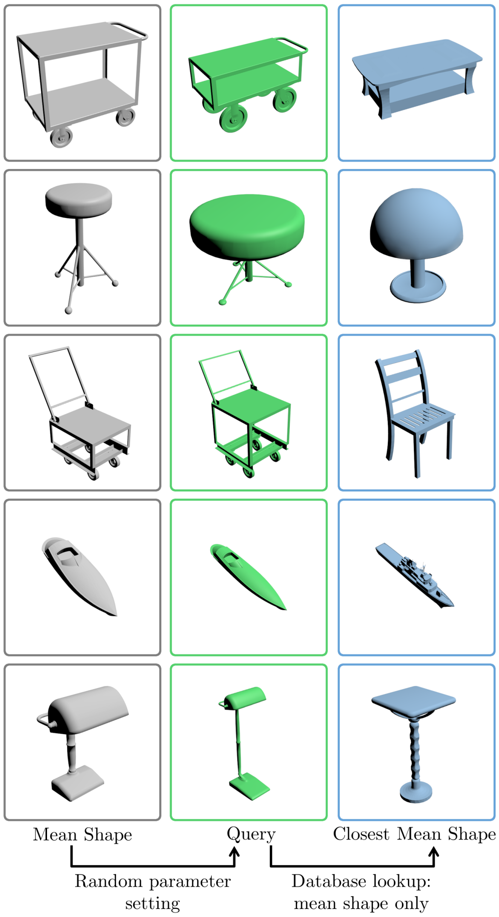


Figure 4-10: Demonstration of query failures when representation only consists of the mean shape. From left to right: the mean shape of some parametric models in the database, query shape given by a random parameter setting of each parametric model, and the closest mean shape retrieved from the database. Since changes in parameter settings significantly alter the geometry, the closest mean shapes are usually not from the parametric models that originate the queries.

targeted errors, δ. Results for additional query shapes are included in the supplemental material. The retrieved results are shown with the parameters settings of the closest primitive. In the case of tangent primitives we simply use the center of the ellipsoids and no additional projection. As discussed in Section 4.5, an additional fitting step would be required to select the optimal parameter configuration.

The quality of our retrieval results is obviously heavily dependent on the descriptor (Figure 4-12). For example, while the Voxel descriptor visually outperforms the D2 descriptor with chairs, the D2 descriptor appears to do better at retrieving lamps. As expected, the Light Field descriptor retrieves better results in every category. The figure also illustrates results for different target errors. Using measurements based on the descriptor space metric, we confirm that with smaller target errors the retrieved shapes are closer to the query. However, the relationship between the visual similarity and proximity in descriptor space highly depends on the quality of the descriptors. For example, using the D2 and Voxel

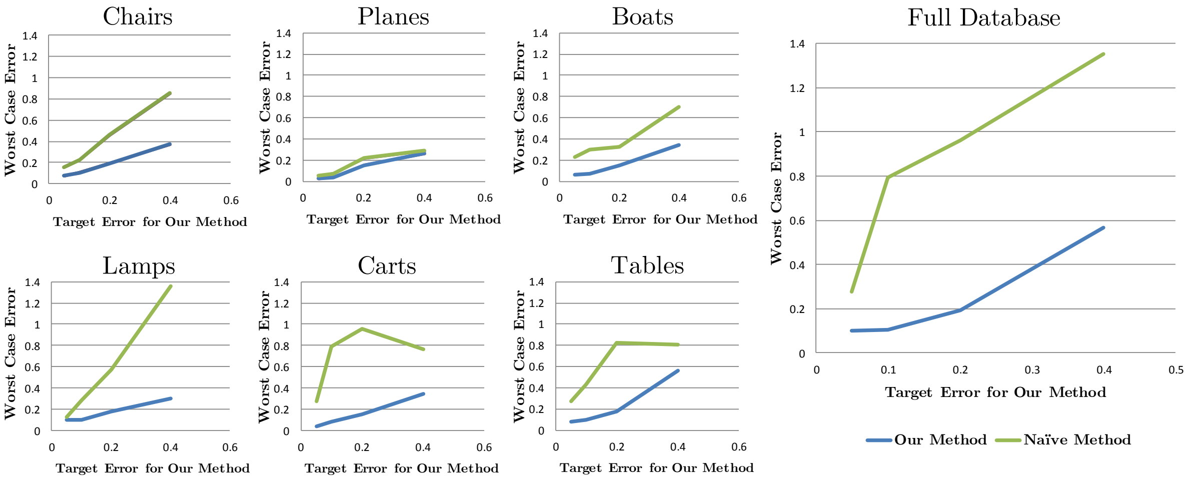


Figure 4-11: Comparison between our approach and the naive one. We measure the difference between the distance to the closest primitive in the collection and the distance to the correct manifold and show the worst case for both approaches for querying points sampled on the full database and on individual categories. From these results we verify that our method has a better performance across all categories.

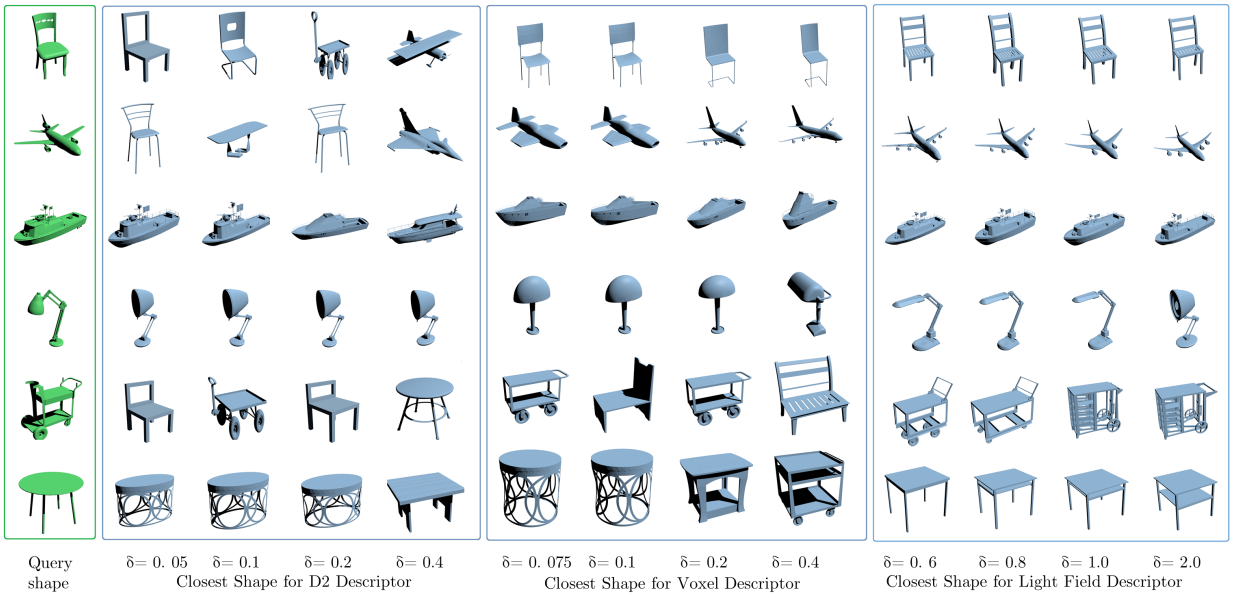

Query shape
δ= 0. 05
δ= 0.1
δ= 0.2
δ= 0.4
Closest Shape for D2 Descriptor
δ= 0. 075
δ= 0.1
δ= 0.2
δ= 0.4
Closest Shape for Voxel Descriptor
δ= 0. 6
δ= 0.8
δ= 1.0
δ= 2.0
Closest Shape for Light Field Descriptor

Figure 4-12: Results of retrieval. From left to right: query shape, result for D2 descriptor (for increasing target errors), and result for Voxel descriptor (for increasing target errors) and results for the Light Fied descriptor (for increasing target errors).

descriptors, smaller δ's result in shapes that are visually less similar for the airplane query example. Nevertheless, we argue that since the retrieved results are actually closer according to the descriptor metric, better descriptors will retrieve more visually accurate results. In fact, this is what happens with the Light Field descriptor.

Classification Searching in the space of parametric shapes allows capturing structure preserving variations during retrieval. When parametric variations are taken into account using our manifold representation, each shape covers a much larger area on the search space. This change of the search space can affect classification in non-trivial ways and is also very dependent on the choice of descriptor. We analyse the effects in classification using Table 4.2 and Figure 4-13.

Table 4.2 compares the search space for different descriptors when single mean shapes are used and when the full manifolds are represented. The distance between categories is measured as the average of the pair-wise minimal distance between categories. The category size is measured by the maximal distance between two shapes in a category and the average across all categories is shown. This result shows that when parametric representations are used the classes become closer to each other and the space covered by each class becomes larger. This is expected since parametric shapes include structure preserving variations. There are however, significant variations depending on the descriptor. While the average distance between categories is reduced by 93% for the D2 descriptor, the reduction is only 31% for the Light Field descriptor.

Table 4.2: Comparison between coverage regions in descriptor space.

Descriptor	Distance Between Categories		Average Category Size	
	Mean Shape	Manifold	Mean Shape	Manifold
D2	0.16	0.02	0.98	1.83
Voxel	0.40	0.16	1.17	1.54
Light Field	1.75	1.12	3.61	5.37

Figure 4-13 shows the standard precision recall plot, which measures classification accuracy. Curves closer to the horizontal line at precision = 1.0 indicate superior retrieval results. Since classification depends on the descriptor, we notice a clear improvement in performance in the Light Field descriptor when compared to the D2 and Voxel descriptor. This result is consistent with mean shapes and the manifold representation. We notice however that while the manifold representation outperforms the mean shape on the Light Field descriptor, the results are equivalent (or slightly worse) for the other 2 descriptors.

From Figure 4-13 and Table 4.2 we conclude that when low quality descriptors are used, classifiers have poor predictive performance and the additional complexity added by the deformation parameters cannot be captured. Therefore they do not help performance and can even act as noise, increasing the error. However, when high quality descriptors are used, the variations of the parametric representations allow better coverage of the space, improving classification performance.

We emphasize however, that the application of retrieval in parametric shape collections goes beyond classification. This is illustrated in Figure 4-14, which uses the Light Field descriptor and compares the mean shapes and the manifold approximation. Results show that the increased variability in the search allows closer matches to be found. In some cases the retrieved results remain in the same category (see the boat, lamp and cart examples). For

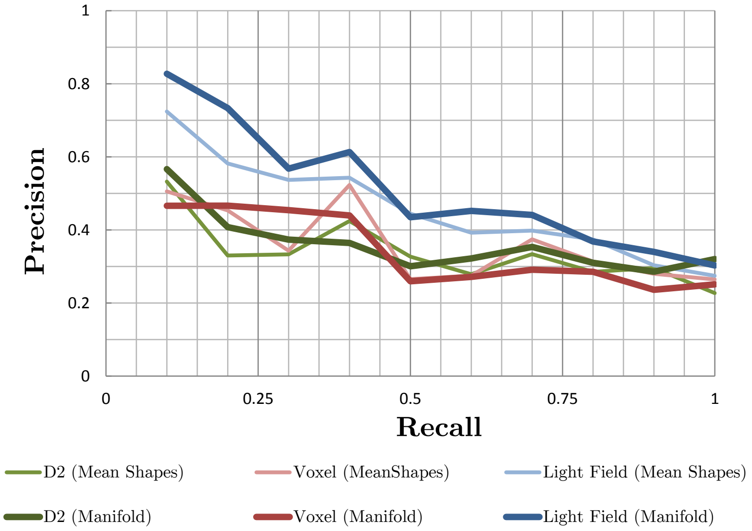


Figure 4-13: Precision-recall plots evaluating classification accuracy for our method compared to using only mean shapes for different descriptors.

the table example, however, the parametric shape space search returns a stool that although belongs to different category (chairs), can be deformed to resemble a table. Though in terms of classification this is an inaccurate result, we notice an improvement in geometric proximity when comparing it to the table retrieved by querying the collection of mean shapes. In the case of a bench query, since we have no database models in this category, the mean shape search finds a boat that has similar dimensions. Our approach, on the other hand, can represent variations of the chair category that make it resemble a bench. This added capability of our technique is not captured by simply using precision-recall classification metrics.

4.7.3 Limitations

Although the main focus of this work is a method to represent a manifold created by parametric shapes in descriptor space, the results of retrieval will always rely on the quality of the actual descriptor. We have tested the retrieval on three different descriptors and observed a large variation in performance. Other descriptors could be tested in our approach as long as they are smooth, i.e., that the region covered by the parameters in descriptor space is close to a manifold.

Another important limitation to discuss is scalability. In our algorithm, storage size is not directly determined by the number of parameters (i.e., dimensionality) but by the volume of descriptor space relative to the tolerance. This volume indicates the variability of valid shapes for a given parametric model, which depends not only on the number of dimensions but also on the ranges of the parameters. For example, one of our airplane models with 8 parameters needs less than a third of the storage of a lamp model with only 3 parameters. Nevertheless, in theory, the volume can increase exponentially with the number of parameters and therefore, our method, like most dimension-dependent representations,

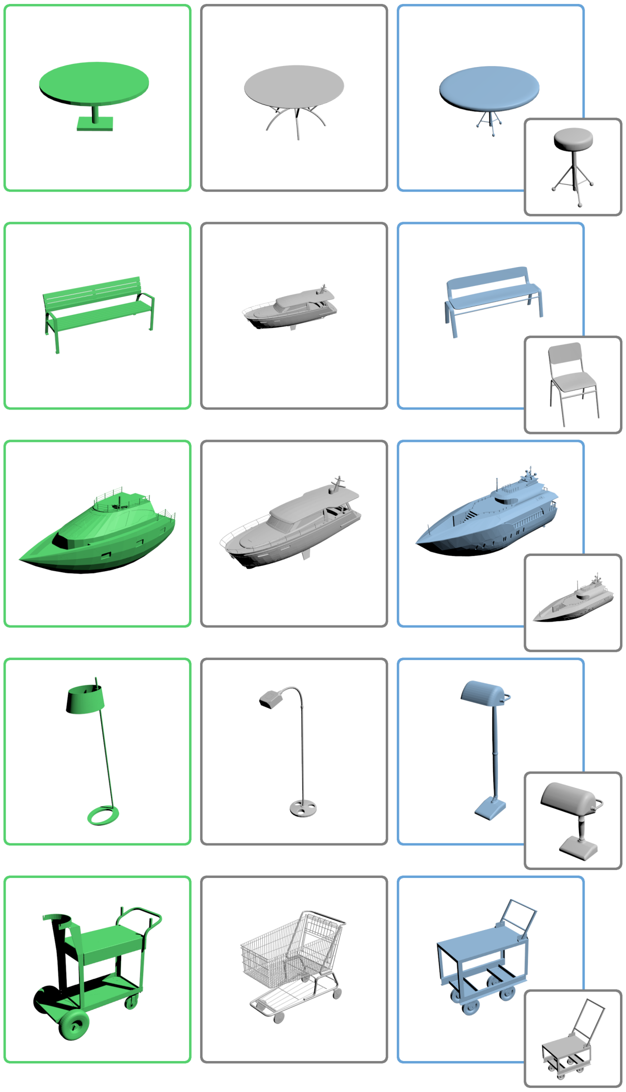


Figure 4-14: Comparison of retrieval with mean shapes only and manifold representation for the Light Field descriptor. From left to right: query shape (green), closest mean shape retrieved (gray), closest parametric shape retrieved with parameter fitting (blue) with its corresponding mean shape (gray). We observe that using the parametric shapes we retrieve models that are more similar in geometry but may lie on a different class.

would not scale. We argue however that, in practice, models with a large number (and volume) of meaningful parameters are not frequently encountered. This is true because although parametric CAD allows for many independent variables, these are often constrained by manufacturing considerations and the need to interface with other models. Therefore the volume of useful variations of a single design tends to be relatively small.

Another assumption that we make is that the feasible set $\mathcal{A}$ is connected. This is mostly relevant for approximating regions of the manifold using tangent spaces and computing boundaries of the ellipsoids. An extension of our work can represent $\mathcal{A}$ as a union of connected sets. It would also be interesting to handle complex boundaries (originated by an arbitrary number of non-linear constraints) as well as a mixture of continuous and discrete parameters. These cases would require more primitives since tangent approximations can only be used on the continuous regions.

Lastly, another limitation of our method is that the time for computing queries scales linearly with the size of the database since we currently use a naive search approach. In high-dimensional descriptor spaces, algorithms based on locality sensitive hashing (LSH) Datar et al. [2004] can solve the nearest neighbor problem in sublinear time. While LSH algorithms typically work with points, one can imagine using ellipsoid centers with LSH to prune obviously far-away regions of the descriptor space and take advantage of such search structures. The effectiveness and feasibility of such methods would need to be tested by experiments.

4.8 Discussion

In this chapter, we present the first approach for efficient retrieval on a collection of parametric shapes that improves upon the standard scheme of first fitting parameters and subsequently computing the distance to the query shape. We address this problem by using shape descriptors and representing parametric shapes as manifolds in this space.

Using a metric for manifold approximation error based on retrieval performance, we propose an algorithm for approximating a parametric shape given a target error. Our approximation consists of a mixture of points and bounded tangent primitives. We discuss how to bound the tangent primitives based on curvature and distance to the boundary. We also define a strategy for optimally selecting primitive types to minimize storage.

Our experiments validate the accuracy of our representation and show that our proposed hybrid representation consistently outperforms approximations that use a single primitive type. Finally, we demonstrate the performance of our method using three different types of descriptors for retrieval on a database of parametric shapes of multiple categories. We observe that the method efficiently retrieves the closest geometry according to the descriptor metric. These may lie outside the original shape categories because of the significant variations imposed by the parametric changes. We envision this approach being particularly useful when shape variations are constrained by manufacturing and for systems that query for parametric parts and then assemble such as the one we will describe in the next chapter of this thesis.

Chapter 5

Assembly-Based Design for Manufacturing

5.1 Introduction

In this chapter, we will describe a composition algorithm that uses the collection of parametric manufacturer designs described in Chapter 3 to create new manufacturable models in a design-by-example manner. This method ensures that the created models can be fabricated, saves the user from many tedious but necessary tasks, and makes it possible for non-experts to design and create actual physical objects (see Figure 5-1).

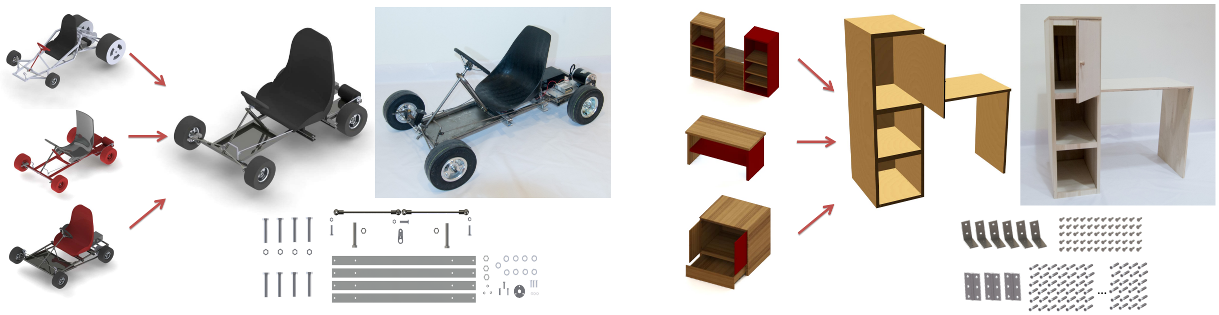


Figure 5-1: The design and fabrication by example pipeline: casual users design new models by composing parts from a database of parametric manufacturable designs. The system assists the users in this task by automatically aligning parts and assigning appropriate connectors. The output of the system is a detailed model that includes all components necessary for fabrication.

Data-driven methods have previously been used to make geometric design easier and therefore more accessible to non-experts. In the "modeling by example" approach, first presented by Funkhouser et al. [2004], new objects are constructed by assembling components of existing objects in a database. This allows even novice users to model complex 3D geometry. However, in creating *manufacturable* designs, several challenges arise that have not been addressed in the previous research. First, all the components in the example database must be manufacturable. Second, any manipulation applied to these components must preserve structure and manufacturability. Third, standard computer graphics techniques such as mesh blending cannot be applied to connect and assemble components. In order to combine parts, these parts must be accurately positioned, and real connectors (e.g., screws or hinges) must be used. The best choice of connectors will depend on the geometric, functional, and material properties of the object. Finally, the resulting model must be structurally sound, so that once it is built it will not topple or collapse.

Using the dataset of parametric designs from Chapter 3 and information extracted from them, we created an assembly-based modeling system for novice users. The user can pick and drag substructures from different designs and add them to a working model. The system guides the user through the *snapping* and *connecting* stages. *Snapping* involves automatically positioning the parts relative to each other, and selecting parameters of the new parts to allow connectivity and alignment. *Connecting* involves automatically selecting the appropriate components and connectors that should be added to hold the parts together. This relieves users from many tedious and complex tasks that are nevertheless necessary for feasible fabrication of the models, allowing them to concentrate on the creative process.

5.2 Design Workflow

Our solution is based on the design-by-example workflow [Chaudhuri and Koltun, 2010, Funkhouser et al., 2004]. Figure 8-3 illustrates our user interface. Icons that link to components of the database are displayed on the left; and the canvas on the right is used to design a new model, henceforth called the *working model*. Users compose parts by dragging them onto the scene, and they can also remove selected parts at any level of the hierarchy.

As we explained in Chapter 3, the components of the database have a hierarchical representation. This allows users to compose models from different designs at different levels, i.e., they can add and remove small components (e.g., a single shelf), medium components (e.g., a drawer), and large complex ones (e.g., an entire cabinet). When creating a new model, they can either start from scratch or work from an existing design.

Users can vary the shape of any component in the database by manipulating its parameters (see Section 5.3). Our system handles composition to ensure that the working model, like the components of the database, is a hierarchical, manufacturable, parametric design. Therefore, users can continue to manipulate parameters after components are assembled.

Composition in a fabrication-aware system is difficult because one cannot merely apply simple geometric operations to merge parts. To combine two substructures, the substructures must be perfectly fitted and aligned, and appropriate connecting elements (e.g., screws and hinges) need to be added between them. This process is not only tedious, but sometimes impossible for users who lack the necessary expertise.

Our system addresses these difficulties with two crucial operational features. First, when a user drags in a new component and drops it onto the scene, the system automatically uses information stored in the database to adjust the component's position and size so that it fits and aligns with the working model. We call this procedure *snapping*. The snapping operation optimizes the component's position and size based on the position in which the user dropped the component and the component's current dimensions. If the user is not satisfied with the

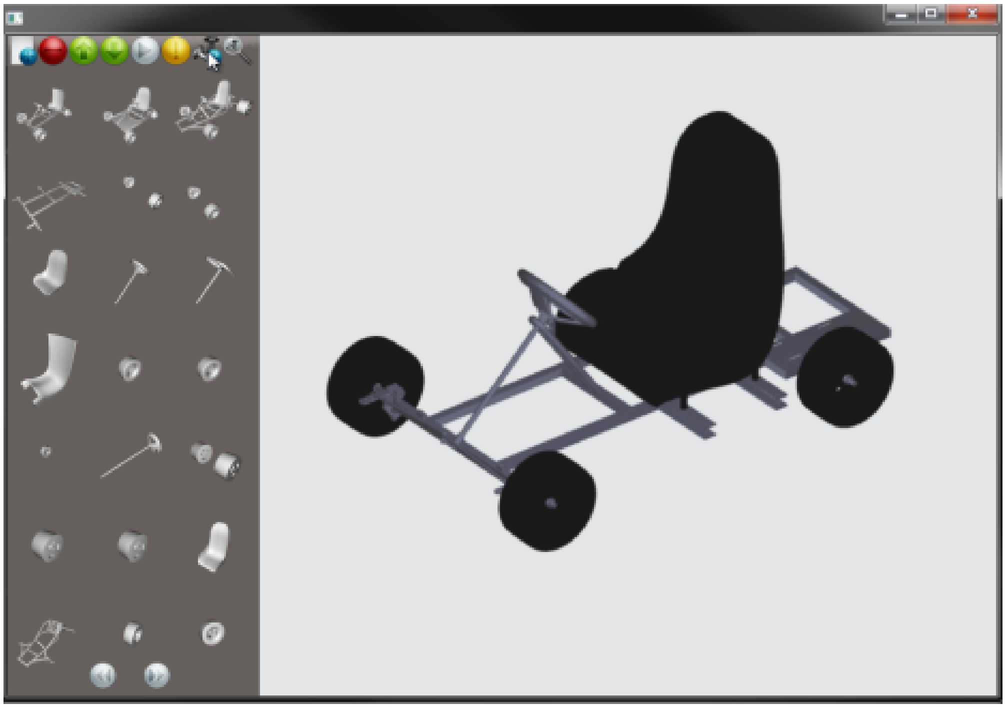


Figure 5-2: The user interface. Icons that link to components of the database are displayed on the left, and the modeling canvas is on the right.

snapped configuration, the user can edit the component (through parameter manipulations) and drag it around the scene, and the snapping procedure will then automatically compute the component's new optimal position and size.

Second, our algorithm automatically retrieves new connecting elements that attach the added component to the working model. This is achieved by searching the database for similar examples of connections. During this process, the system computes new geosemantic relationships between the added parametric part and the working model. Both the new connecting elements and the new geosemantic relationshops are added to the hierarchy of the working model together with the added component. We call this process *connecting*.

5.3 Parametric Manipulations

Allowing users to manipulate parameters adds variety to the designs (see Figure 1-3) and allows fitting parts of different sizes. Adjusting parameters modifies a design's shape and dimensions, but preserves its overall structure. This method of design manipulation guarantees that the composed models are manufacturable.

The system allows parameters to be manipulated at all levels of the hierarchical structure. The user can select elements (leaf nodes) by clicking on them, and then traverse up the hierarchy to select internal nodes. When a node is selected, controls for scaling and translation are revealed (see Figure 5-3). At each level of the hierarchy, the controls act on the bounding box of the selected parametric design T^i. Therefore, the user can make higher-level changes by selecting internal nodes and make more detailed adjustments by selecting leaf nodes.

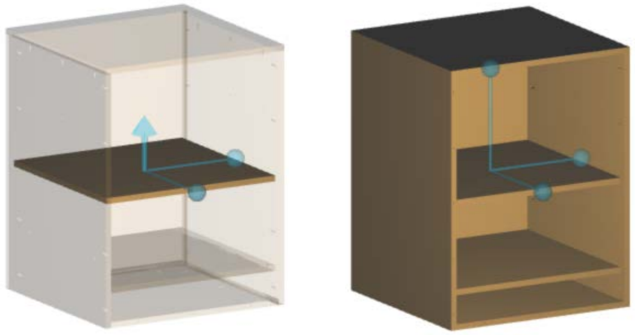


Figure 5-3: An illustration of how parametric variations can be explored in our tool. The arrows control translation, while spheres control scaling. On the left, we show the controls on a leaf node of the hierarchy; on the right, we show the controls on an internal node. During manipulation, elements on the selected node are represented in full color, while the others become semi-transparent. Notice that constrained degrees of freedom are hidden. For example, the user is unable to change the thickness of the shelf, since the items catalog states that planks of wood can be cut only in two directions.

Parameter manipulation is not an unconstrained procedure. A given design is restricted to the feasible space $\mathcal{A}$ stored at the root node of the hierarchy. As outlined in Chapter 3, we define the parameters of the root node $\mathbf{q}$ as the stacked vector of all the children $\mathbf{q}^i$. We can then represent all linear, geosemantic constraints (bilateral and unilateral) in the standard form: $\mathbf{A}\mathbf{q} = \mathbf{b}, \mathbf{G}\mathbf{q} \leq 0$. We augment $\mathbf{A}$ with constraints that fix the center of the model in order to prevent translation of the edited parametric model.

To create the controls we use six functions $\mathbf{c}_j$, such that the $\mathbf{c}_j^T\mathbf{q}$ correspond to the center and dimensions of the axis-aligned bounding box. When manipulating leaf nodes, the $\mathbf{c}_j$ are standard basis vectors, since $\mathbf{q}^i$ defines the axis-aligned bounding box of the element. For larger substructures represented by internal nodes, we first compare the bounds of the children elements to determine which ones constrain the bounding box of the substructure in each dimension. We can then use these elements to explicitly determine the functions $\mathbf{c}_j$.

As the user drags a control j, we calculate the new parametric configuration by solving the simple quadratic program

$$\begin{aligned} \mathbf{q}^* = \underset{\mathbf{q}}{\operatorname{argmin}} \|\mathbf{c}_j^T\mathbf{q} - (\mathbf{c}_j^T\mathbf{q}_{\text{current}} + \delta)\|^2 + \alpha\|\mathbf{q} - \mathbf{q}_{\text{current}}\|^2 \\ \text{s. t.} \quad \mathbf{A}\mathbf{q} = \mathbf{b}, \mathbf{G}\mathbf{q} \leq 0 \end{aligned} \tag{5.1}$$

where $\mathbf{q}_{\text{current}}$ are the parameters in the current state and δ determines the amount of dragging. The second term penalizes large changes in parameter value. Accordingly, α is chosen to be less than one to give more importance to the first term. To reduce cluttering, we hide controls that manipulate a completely constrained scaling direction. We determine whether the j^{th} control is constrained by checking if $\mathbf{c}_j$ and $\mathbf{A}$ are linearly dependent.

5.4 Composition

We compose new designs by removing and adding components to the working model. To remove a part, the user explores the working model's hierarchy and selects a substructure of the model. The corresponding node is excised from the tree. The hierarchical nature of our representation is leveraged to quickly remove all connections and geosemantic relationships incident on the deleted structure. This frees the working model of unnecessary constraints and connecting elements that serve no purpose in the new design.

As mentioned earlier, adding new parts is more difficult; therefore, our method assist the user in two ways. First, when a user chooses a parametric part T^A, we adjust the dimensions and placement of T^A to *snap* it to its position. Second, we automatically compute new constraints and find the elements that *connect* T^A to T^W.

To define both snapping and connecting, our algorithm first examines the original design T^D from which the part T^A originated. It examines where and how T^A connected to T^D, and tries to use that information to align and connect T^A to T^W. If the information it has is not sufficient, it searches the rest of the database for similar connections. In the following two subsections, we explain the snapping and connecting operations more fully.

5.5 Snapping

Snapping the additional part T^A to the working model involves computing a new parametric configuration $\mathbf{q}^A$, which is optimized based on the user's current positioning and dimensions of the part (i.e., the current state of the parameters $\mathbf{q}^A_{\text{current}}$). If the user is not satisfied with the solution, she can continue to change T^A's position or its parameters to bring them closer to the desired configuration, and the system re-optimizes $\mathbf{q}^A$ given the new current state.

To find an optimal parametric configuration, our algorithm tries to identify constraints that $\mathbf{q}^A$ will have to satisfy when T^A is added to the working model. We do this in two steps. First, we analyze how T^A connects to the original design T^D and try to find constraints on $\mathbf{q}^A$ that would allow T^A to connect to T^W in a similar manner. Second, we search for prominent planes on T^A that are sufficiently close to planes in T^W and try to align them. In what follows we discuss each of these steps in detail.

Constraints Based on Original Design. To create constraints based on the original design, our algorithm looks at the elements (leaf nodes) of T^D that are connected to T^A and extract the coplanarity relationships between these elements and T^A. As mentioned in Chapter 3, coplanarity relationships constrain prominent planes—in this case a plane of T^A with respect to a plane of T^D. To create an analogous constraint between T^A and T^W, we need to find a plane on T^W that has the same normal as the one in T^D. Since there might be many planes in T^W that satisfy this requirement, we take the K ones that are closest to T^A in the current configuration.

Using the parametric representation, we can write each connectivity relationship as a linear constraint $\mathbf{a}\mathbf{q}^A = b$, where b depends on the selected plane in T^W. We then extract a subset of the linearly independent constraints $\mathbf{a}$ that is also not restricted by the feasible set $\mathcal{A}^A$ of T^A. Notice that the number R of constraints in this subset cannot exceed the

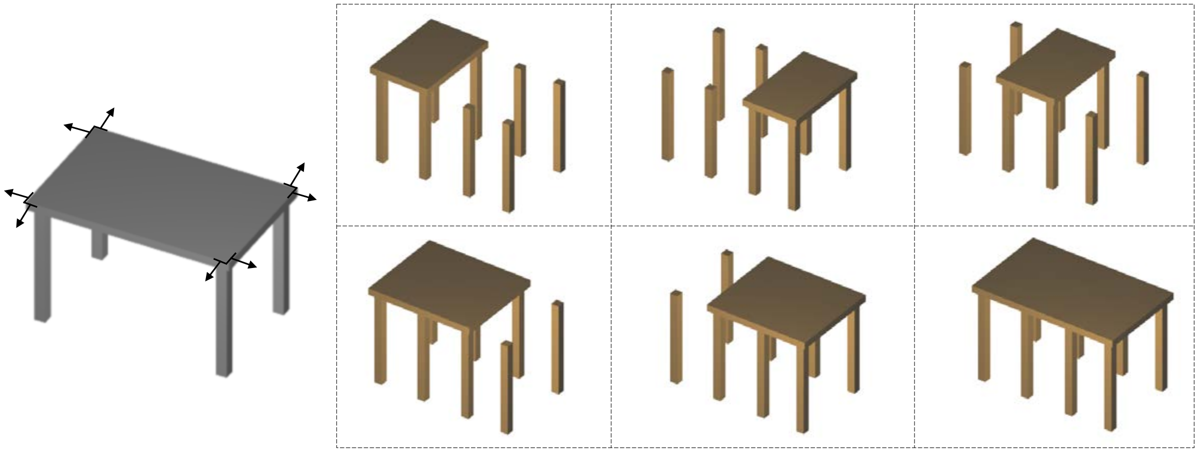


Figure 5-4: An example of snapping to constraints. We add a tabletop T^A to the working model T^W containing eight legs (right). The coplanarity constraints on the original design T^D that contained T^A are represented by the normals of the corresponding planes (left; we show only the vertical ones). The feasible snapping configurations for $\mathbf{q}^A$ are shown on the right. The system will choose one of these configurations: its choice will depend on the scale parameters and the position on which the user places the tabletop.

number of degrees of freedom of T^A, which tends to be small (usually six). Since b can be chosen in K different ways, we end up with a set of $N = K^R$ possible constraints $\mathbf{A}\mathbf{q}^{\mathbf{A}} = \mathbf{b}_n$ that restrict the parameters of T^A. For each of these possible constraints, we compute the optimal $\mathbf{q}^A$ by solving the following least squares problem:

$$\min_{\mathbf{q}^A} \|\mathbf{q}^A - \mathbf{q}^A_{\text{current}}\|^2 \quad \text{s. t.} \quad \mathbf{A}\mathbf{q}^A = \mathbf{b}_n, \quad \mathbf{q}^A \in \mathcal{A}^A \tag{5.2}$$

We then select the constraint matrix $\mathbf{A}\mathbf{q}^A = \mathbf{b}_{\bar{n}}$ whose optimal solution has the smallest cost. The value of K is chosen depending on R to guarantee that $N = K^R$ does not become too large. Typically, we set $K = 4$.

Alignment of Prominent Planes In many cases, the constraints of the original design are not enough to position the part in the working model. For example, in the working model shown in Figure 5-4, the table legs were created by composing and snapping two sets of four legs, which are aligned even though they are not connecting and there is no resemblance to such a combination in the original design of either set. To find additional alignment constraints, we select the set of planes in $\mathbf{T}^A$ that are not restricted by $\mathbf{A}\mathbf{q}^A = \mathbf{b}_{\bar{n}}$. For each of these planes, we find the closest parallel plane in the working model, and we align them if the distance between them is smaller then a certain threshold. This gives us a new $\mathbf{q}^A$ that will be used to connect the additional part to the working model. For objects with functionality (i.e., several configurations), we construct prominent planes corresponding to all main rest configurations (see Appendix 3.5.1). This guarantees that functional objects snap so that they align to the working model in all rest configurations (see Figure 5-5).

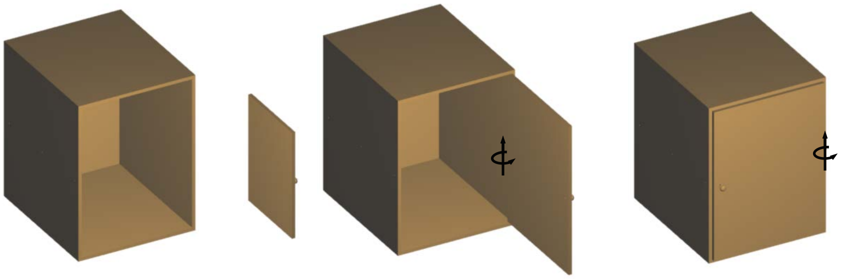


Figure 5-5: An illustration of snapping for functional objects. When a door is added to the side of a cabinet, it automaticaly rescales so that, when shut, it will align with the oposite side. At left, a door is added to the working model. From left to right: the added door before snapping, the snapped configuration, and a visualization of the snapped configuration when the door is closed. The rotation axis of the articulations is depicted by the arrows.

5.6 Connecting

Once the user is satisfied with the fitted part, they invoke the *connecting* method. Connecting automatically places T^A in the hierarchy of the working model and adds the appropriate connections and geosemantic relationships. Although the parameters $\mathbf{q}^A$ may still vary, the snapping result returns an approximate configuration of T^A. We use this information to find the elements in the working model T^W that should be connected to elements T^A based on proximity. We call these *linked elements*. We then search the database for a connection that can be used to connect each pair of linked elements. After these connections are selected, a final composition step is performed to create a new working model that preserves the hierarchical structure and is correctly parametrized. We discuss each of these steps in the following paragraphs.

5.6.1 Searching for Connections

As in our snapping algorithm, we first search for connections in the original design T^D. We consider all the connections in T^D that connect T^A to elements in $T^D \setminus T^A$, and we try to transfer these connections to the working model. Transferring involves matching principal elements in $T^D \setminus T^A$ to elements in T^W. Since we have the $\mathbf{q}^A$ that resulted from the snapping algorithm, we can first fit T^D by finding $\mathbf{q}^D$ that minimizes $\|\mathbf{S}\mathbf{q}^D - \mathbf{q}^A\|$, where $\mathbf{S}$ is a matrix whose columns are the standard basis vectors that correspond to the indices of $\mathbf{q}^D$ that refer to $\mathbf{q}^A$. Once the fitting is done, we can use a standard distance function on the bounding boxes of each element to retrieve the closest matches. Finding the matches creates a candidate connection that acts on $T^W \cup T^A$.

Once we have candidate connections, we need to determine if they can be used in the composed design. A connection contains the set of relationships between the elements it references (see Section 3.6), which can be represented by the feasible set $\mathcal{A}^C$. We cannot add a connection if the feasible set $\mathcal{A}^C \cap \mathcal{A}^A \cap \mathcal{A}^W$ is empty. If the feasible set is nonempty, the algorithm finds the configuration of the composed model $T^W \cup T^A$ in this set that is as close as possible to the snapped configuration. It does this by solving a quadratic program:

$$\begin{aligned} \min_{\mathbf{q}^A,\mathbf{q}^W} & \|\mathbf{q}^A - \mathbf{q}^A_{\text{current}}\|^2 + \|\mathbf{q}^W - \mathbf{q}^W_{\text{current}}\|^2 \\ \text{s. t.} \quad & \mathbf{q}^A \in \mathcal{A}^C \cap \mathcal{A}^A, \quad \mathbf{q}^W \in \mathcal{A}^C \cap \mathcal{A}^W \end{aligned} \tag{5.3}$$

We allow the error to be larger than zero because, in many cases, principal elements need to be slightly shifted or scaled in order to insert connecting elements (see Figure 5-6). Nevertheless, in order to guarantee that we do not drift too much from the user's design, we use a connection only if the error of this minimization is smaller then a fixed threshold.

When not all connections are found in T^D, the algorithm extends the search to all designs in the database. We use a priority based on a similarity metric that compares connections by evaluating: 1) similarity of principal components, and 2) the relative distance between them. We compare principal components by first making sure that the materials match and then measuring the distance between the sizes of the bounding boxes. Since the dimensions of the components can vary according to parameter manipulations, we would like the similarity metric to encode a weighted average of the amount of parameter manipulation necessary and the final distance between the two components. We accomplish this by giving extra weight to dimensions that are constrained, which naturally encode the distance between the models after fitting.

We compare relative distances between elements using the distance between the bounding planes of the axis-aligned bounding boxes of each element. For each dimension, we compare both bounding planes against each other—a total of four evaluations. By doing so, we encode not only coplanarity relationships but also order. This is important because parts often need to have an intersecting area in order to be connected.

We pre-compute the descriptors for all the connections in the database, so that a simple weighted distance function efficiently retrieves the closest candidate connections at runtime. After the candidate connections are retrieved, we evaluate them using the method described above. We try only the K closest connections: if none of them pass the evaluation, we refrain from adding a connection and warn the user that no connection was found. Typically, we set $K = 5$.

5.6.2 Final Composition

Once we retrieve the set of edges, we am ready to generate the new working model $T^{\bar{W}}$ that incorporates T^A into T^W. We place T^A into the hierarchy of T^W by adding it as a sibling to the lowest node that groups the elements that connect to T^A. We add all the connections found by transferring the connecting elements to the working model and adding the relationships between them and the elements of $T^{\bar{W}}$. These relations constrain $\mathbf{q}^A$. Once these constraints are built, we optimize $\mathbf{q}^{\bar{W}}$ so that it is as close as possible to the current snapped configuration (solving a least squares problem). This may effect minor changes in the parameters of T^A. For instance, the parameters may change to allow connecting elements to fit between parts, as in Figure 5-6. Finally, we find and add geosemantic relationships between elements of T^A and T^W automatically in the same manner in which we built the original parametric designs (see Section 3.4).

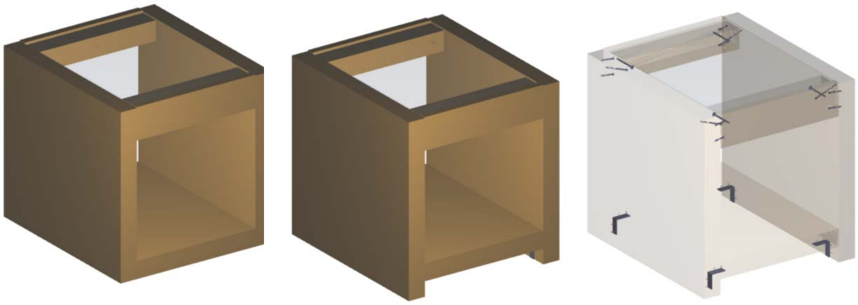
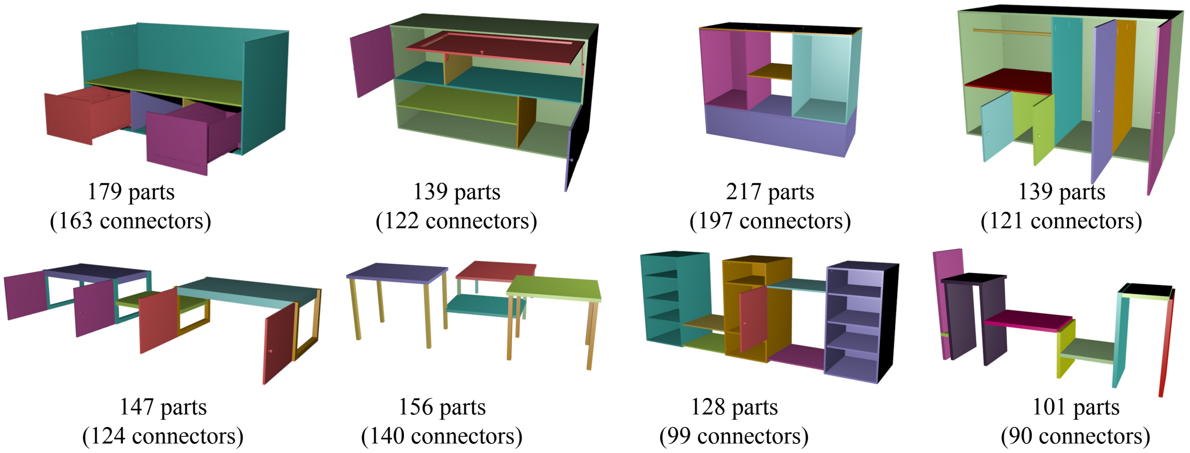


Figure 5-6: An example of changing parameters to fit connectors. From left to right: the bottom shelf snapped to the bottom of the table, the resulting configuration of the model after the connecting step, and the vizualization of the connectors (principal elements are made semi-transparent). Notice that, in order to connect the bottom shelf to the table legs, the system raises the shelf above the ground to leave room for l-brackets.

5.7 Results

5.7.1 Modeling



179 parts
(163 connectors)

139 parts
(122 connectors)

217 parts
(197 connectors)

139 parts
(121 connectors)

147 parts
(124 connectors)

156 parts
(140 connectors)

128 parts
(99 connectors)

101 parts
(90 connectors)

Figure 5-7: Examples of models designed using our system and the number of individual parts they comprise. Different colors indicate the different parts that were added to the model.

Figure 8-11 depicts a few results built using our tool. We indicate in the figure the number of individual parts in each design. Observe that even models that appear to be simple are composed by over one hundred parts. It would take an expert from one to four hours to build each of these models with commercial CAD software. However, using our system, users with no expertise in mechanical engineering were able to create these designs in less than twenty minutes.

In Figure 8-11, we highlight the different parametric designs that were added to the model using different colors. Notice how the number of elements in each color-coded design varies.

This illustrates how the users can explore the hierarchy by composing parts using smaller or larger substructures.

While the snapping and connecting steps of our system are responsible for the speed in which users can create such complex objects, the parametric manipulation feature helps to add diversity to the models. By adding new geosemantic relationships each time parts are added to the working model, we guarantee that the working model maintains the parametric representation which allows structure–preserving manipulations. Figure 5-8 shows an example of how the user can continue to explore the space of parameter variations of a composed model.

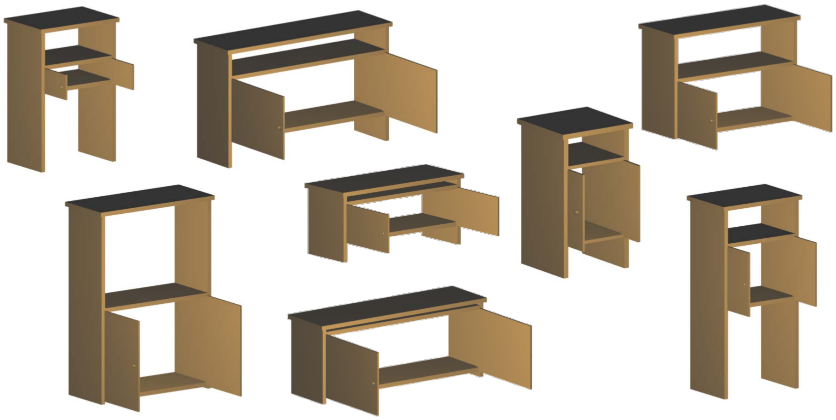


Figure 5-8: An example of different manipulations of a working model after it has been composed from multiple parametric designs.

5.7.2 Fabrication

We tested the full data-driven fabrication pipeline to build four designs, illustrated in Figures 4-1 and 5-9. These models were created by combining parts from multiple designs, also shown in the figures. The output of the system is a comprehensive bill of materials that is generated by looking up the items for each part in the items catalog. Then, using the information provided by the external supplier, users easily order the items and then assemble them. It is also possible to minimize the total cost of materials by grouping together and combining items – for instance, by cutting many wood elements from a few pieces of stock material.

5.8 Discussion

In this chapter we presented what we believe to be the first complete data-driven system for digital fabrication. Our algorithm successfully leverages a database of parameterized fabricable designs, allowing casual users to design models that can be physically realized. The output of our algorithm is comprehensive in that it provides a list of all parts necessary for construction, as well as a detailed bill of materials that lists where parts can be purchased and the total cost of construction. We have demonstrated the power of our method by fabricating

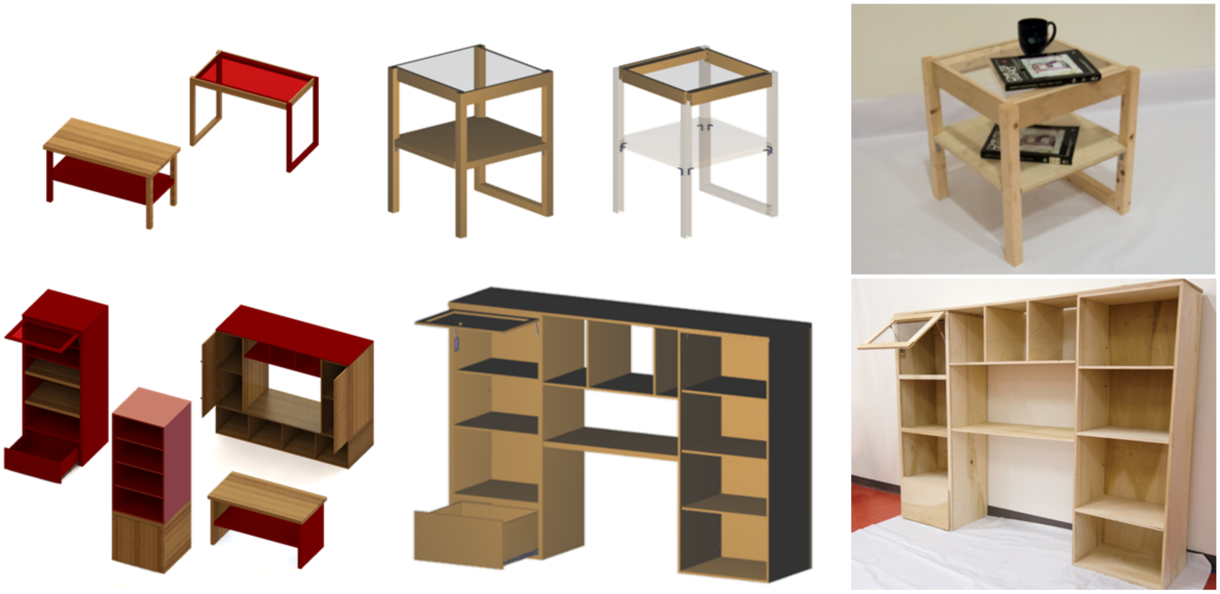


Figure 5-9: From left to right: input designs, models created using the system, and fabricated results. We highlight the connecting elements on the first model by making all principal elements semi-transparent.

different models. We have shown the scope of the data-driven method by applying the same algorithm to fabricate furniture and go karts.

In this chapter, however, we only show results on geometric composition. Although we can handle some functionality like drawers, cabinet doors, and wheels, the composition algorithm cannot handle more complex dynamic components such as electronics. In the go kart examples shown in the teaser, the algorithm was able to compose wheels (adding bearings), the seat (adding connecting planks with bolts) and the steering (adding in tierods). However, it could not interchange motors and controls, and so we considered them as part of the frame assembly of the go kart. Systems that can simultaneously design both geometry and motion/control are covered in Chapter 8.

In order to build such systems, we need further analysis on how the objects will behave once manufactured. While the data-driven algorithms we have discussed so far can be used to constrain the design space to ensure manufacturability—every model created with this method has a corresponding bill of materials—, nothing can be said about how well the resulting designs will *perform* once they are part of the physical world. Performance-driven design algorithms which allow efficient exploration and optimization of the design space driven by performance metrics is the topic of the next two chapters of this thesis.

Chapter 6

Interactive Design-Space Exploration

6.1 Introduction

In the previous three chapters, we discussed data-driven methods that allow constraining the design space in meaningful ways—we use parametrization to define degrees of freedom for design variability preserving manufacturability. As shown in Figure 1-2, the next question we should address is, how can we optimize this (now, constrained) design space driven my performance evaluations?

In this chapter we will focus on the forward problem of mapping design space to performance space. We will use generic CAD models for defining the design space, since CAD systems are used by almost every mechanical engineer in the world to create practically every existing manufactured shape. CAD models are parametric from construction and capture the engineer's design intent, including manufacturability. In a typical engineering design application, designers will expose a small set of CAD parameters over which they optimize for performance. However, since they need to evaluate a large number of points in design space, it typically takes up to a day for a mechanical engineer to select parameters for a complex shape.

To address this problem, we propose a new algorithm that allows interactive exploration of the design space defined by CAD parameters. As shown in Figure 6-1, the output of our method is a simple interface where users can explore the parameter space with real-time feedback on performance. Computing this feedback in interactive rates is challenging because evaluation of geometry for CAD parameter variations is inherently slow (see Section 2.1.2). In addition, performance evaluations typically involve time-consuming physics simulations—e.g., FEA methods for stress analysis, which can take 5-10 minutes for a high resolution model).

We address these challenges by offloading to a precomputation stage the mapping from parameters to geometry. Because different parameters can influence the geometry in different ways and to different extents, we sample the parametric domain using an adaptive grid. Interpolating in this space is challenging because each sample is a mesh with different combinatorics— parameter changes result in different B-reps (Boundary Representation) with incomplete correspondence and generate meshes with different number of triangles/tetrahedra. Moreover, interactive exploration applications require continuity across the inter-

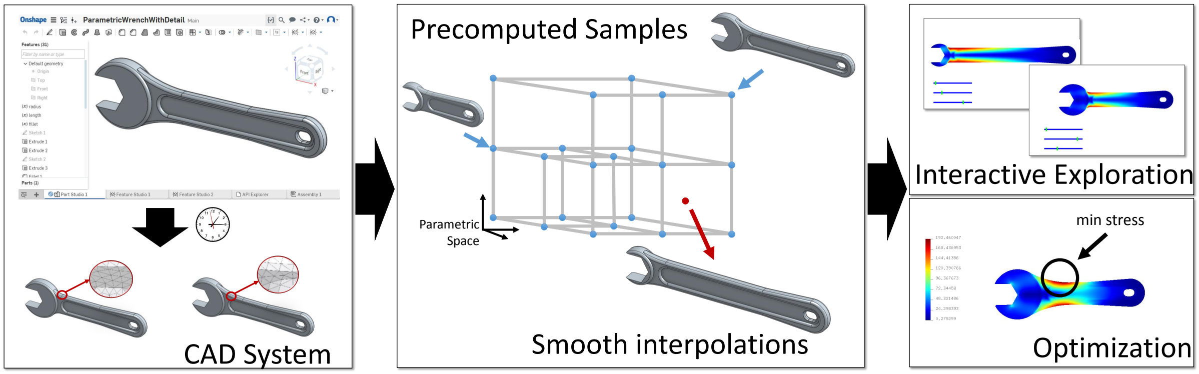


Figure 6-1: Our method takes as input a CAD model with a set of exposed parameters which define the design space. We sample the parametric space in an adaptive grid and propose techniques to smoothly interpolate this data. We show how this can be used to drive interactive exploration tools that allow designers to visualize the shape space while geometry and physical properties are updated in real time.

polated domain.

To address these problems, we propose a method that takes advantage of the underlying CAD representation (B-reps) to compute correspondences between shapes that are close enough in parameter space. This makes our interpolation possible, but with the caveat that, to keep the computations tractable, we must restrict the influence of each sample to local regions. But constructing an interpolation that both satisfies this *locality* property on a non-uniform domain and is *continuous/smooth* is a real challenge.

The main technical contribution of our work is a novel interpolation algorithm on adaptive grids that overcomes this difficulty, allowing for approximations across the entire parameter space that are *both* smooth *and* local. The proposed method is a generic interpolation scheme that can be applied independently of the dimensionality of the domain and can be combined with different types of basis functions (e.g. linear and cubic B-splines).

6.2 Related Work

Adaptive grids have been extensively used to efficiently store non-uniform data. The challenge for interpolating in these domains comes from the hanging nodes (also called T-junctions) at the boundary between elements at different levels. Approaches to handle discontinuities around the hanging nodes include temporarily subdividing elements [Benson and Davis, 2002, Kobbelt et al., 1997], constraining T-junction nodes on edges to be linearly interpolated from their neighbors on that edge [Agarwala, 2007, Losasso et al., 2004, Xu et al., 2009], using penalty functions to minimize discontinuities [Setaluri et al., 2014], or introducing generalized barycentric coordinates [Floater, 2015, Sukumar and Malsch, 2006]. T-Splines generalize NURBS surfaces to accommodate T-vertices [Sederberg et al., 2003]. All of these approaches have their drawbacks: they either come at the expense of generating many virtual nodes and further subdivision, lose data samples, do not guarantee continuity, are hard to extend to high dimensions, or are not easily extended to smooth interpolations.

The solution for smooth interpolations is the use of hierarchical basis functions [Lee et al., 1997]. But this comes at the price of locality: the value of a point at a given cell depends on samples that lie outside the boundary of that cell. The particular nature of our problem stems from the fact that, though we require continuous interpolations for exploration applications and smooth interpolations for derivatives in optimization problems, we also have a strong restriction of locality because averaging between samples involves morphing between meshes. Our method addresses these challenges using refinable basis functions to construct interpolants in each sampled node. We describe a refinement algorithm that updates these interpolants as elements of the K-d are refined, thereby guaranteeing locality. This algorithm is simple and easy to implement in K-d trees of any dimension or grading. Most importantly, it can be implemented with multiple basis functions, allowing not just continuous, but *smooth* approximations.

6.3 Workflow

The workflow of our system is shown in Figure 4-1. In an offline phase, CAD shapes are precomputed by adaptively sampling the parametric domain. Interpolating basis functions for each sample (Section 6.5) and compatible meshes for interpolation (Section 6.6) are computed during this phase. For some applications, in addition to storing the geometry at each sample, we also precompute and store different simulation results.

For exploration, we use the typical user interface with sliders for interactively modifying shape parameters (see top-right corner of Figure 4-1). Both geometry and precomputed physical properties are interpolated in real time and displayed. For this application, we use a continuous interpolation function to avoid flickering when switching between regions of the adaptive grid.

6.4 Precomputation Overview and Notations

A parametric shape is defined as a function that returns a geometry for each parameter configuration $x \in \mathcal{A}$, where $\mathcal{A}$ is the feasible parameter set. Here and henceforth we assume that a sample value p_k at $x_k \in \mathcal{A}$ can be evaluated by interfacing with a CAD system and consists of a tetrahedral mesh. Our method computes an approximation $P(x)$ of the parametric shape by interpolating these samples.

We use a K-d tree in parameter space for sampling and interpolation (see Figure 4-1). We use the term *element* to refer to the cells of the K-d tree. The parametric shape is sampled at every corner of every element. To each sample x_k we associate a *basis function* ψ_k and approximate the shape by interpolating the samples:

$$P(x) = \sum_k p_k \psi_k(x). \tag{6.1}$$

Definition 1 (Support)**.** *The support of each basis function $S(\psi_k)$ is defined as region on which it assumes non-zero values.*

The challenge with computing the above sum is that p_k are meshes with varying combinatorics. To address this, let us assume a canonical mesh p^l for each element e_l. Then, if $x \in e_l$, we can compute $P^l(x)$ by converting each sample x_k whose support contains e_l to this canonical mesh p^l_k:

$$P^l(x) = \sum_{k, e_l \in S(\psi_k)} p^l_k \psi_k(x). \tag{6.2}$$

To keep this problem tractable, we would like to limit the support of the basis functions. To this end, we make the following definitions.

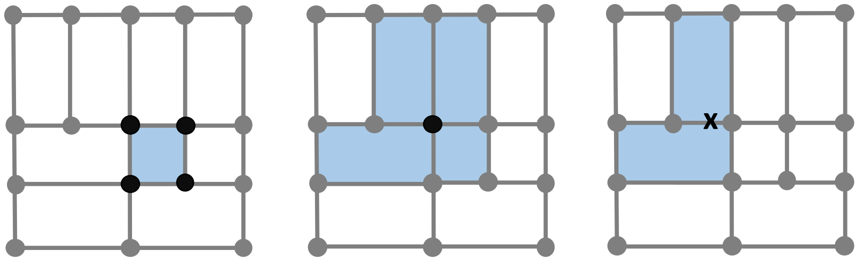


Figure 6-2: The neighborhood of an element, denoted $\mathcal{B}(e_l)$, is defined as the set of adjacent samples (left). The neighborhood of a *sample*, denoted $\mathcal{N}(x_k)$, is defined as the set of adjacent elements (middle). We also extend the definition of $\mathcal{N}(x_k)$ to any point $p \in \mathcal{A}$ as the set of adjacent elements (right).

Definition 2 (Element Neighborhood)**.** *The neighborhood of an element* $\mathcal{B}(e_l)$*, is defined as the set of samples on its boundary (Figure 6-2).*

Definition 3 (Sample Neighborhood)**.** *The neighborhood of a sample* $\mathcal{N}(x_k)$ *is defined as the set of elements incident to the sample* x_k*. This definition can be extended to any point* $p \in \mathcal{A}$*, where* $\mathcal{N}(p)$ *is the set of elements that contain or are incident to* p*. (Figure 6-2).*

Definition 4 (Locality)**.** *We say that an approximation preserves locality if* $S(\psi_k) \subset N(x_k), \forall k$*. Analogously, if an approximation preserves locality, then for any sample* x_k *and element* e_l*, then* $e_l \subset S(\psi_k)$ *only if* $x_k \in \mathcal{B}(e_l)$

From these definitions, when an approximation preserves locality, we can rewrite equation 6.2 as:

$$P^l(x) = \sum_{k, x_k \in \mathcal{B}(e_l)} p^l_k \psi_k(x). \tag{6.3}$$

This implies that only samples $x_k \in \mathcal{B}(e_l)$ need a mesh p^l_k that is consistent with this element (Figure 6-3).

In order to guarantee that there are no discontinuities when crossing between elements, we would like $P(x)$ to be continuous. From Equation 6.1, we see that it is enough to ensure that ψ_k is continuous for all samples x_k.

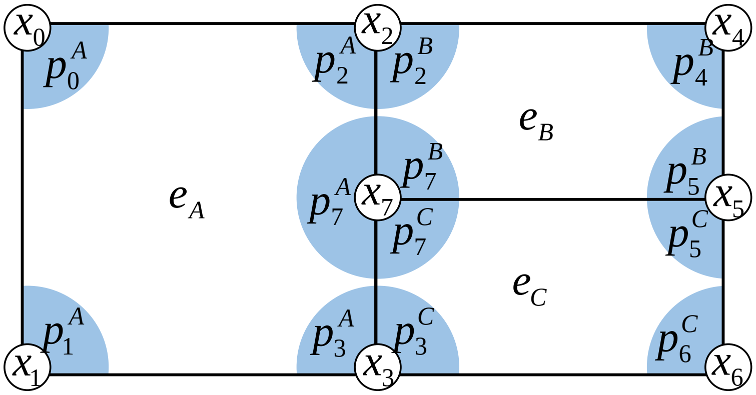


Figure 6-3: When locality is enforced, the number of consistent representations p_k^l that are needed at each sample x_k depends on the cardinality of $\mathcal{N}(x_k)$.

6.4.1 Refinement Relations

To propose a scheme that guarantees locality and continuity on an adaptive grid, we take advantage of basis functions that observe a *refinement relation* which we define now for notation purposes. A refinement relation allows for expressing a basis function as the sum of dilated translated versions Zorin and Schroder [2000]. In our algorithm we use linear B-splines, illustrated in Figure 6-4. A basis function from a coarser level can be written as a linear combination of basis functions from the next finer level:

$$\phi_i^j(x) = \sum_n a_{in}^{(j+1)} \phi_n^{(j+1)}(x), \tag{6.4}$$

where the superscript j indicates the level of refinement ($j = 0$ corresponding to the original, coarsest grid), and the subscripts i and n index the basis functions at the respective levels. In the case of linear B-splines, the coefficient a_{in} do not depend on j and are given by

$$\phi_i^{(j)}(x) = \frac{1}{2}\phi_{(2i-1)}^{(j+1)}(x) + \phi_{(2i)}^{(j+1)}(x) + \frac{1}{2}\phi_{(2i+1)}^{(j+1)}(x). \tag{6.5}$$

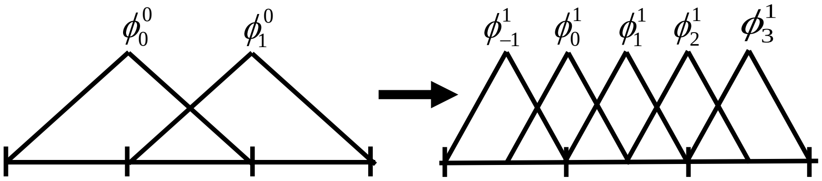


Figure 6-4: Refinement of linear B-splines.

6.4.2 Adaptive Sampling

We build the K-d tree as follows. The parametric domain defines an element e_0 at the coarsest level where every corner vertex is sampled. We associate to each sample x_k a single B-spline at the coarsest level, $\psi_k = \phi^0_k$.

We establish a canonical mesh p^l for each element by evaluating the geometry at the center of the element and using the mapping algorithm (Section 6.6) to establish a consistent meshing inside each element. We store the consistent meshes for each of the samples on the cell boundary p^l_k (see Figure 6-3).

We initially refine all samples once in each direction and then continue to refine adaptively. At this stage, to decide whether an element e_l needs refinement, we evaluate the application-specific approximation error at the center of e_l and refine it if this value is above a given threshold. Since we have consistent meshes, the error above can easily be computed with application-specific metrics (see Section 6.7). To decide the refinement direction on a high-dimensional K-d tree, we compute the distance in each direction and use the maximum, again taking advantage of the consistent meshes. This process is similar to the work of Shugrina et al. [2015].

Our method iterates over the leaf nodes of the K-d tree checking which need to be refined. We use a priority queue with the size of the element as priority. When an element e_l is split, its children are added to this queue. Also, since this refinement also affects the interpolated result on the elements adjacent to the split element e_l, these are also added to the queue.

6.4.3 Refinement Notations

When an element is refined, new samples are added along the split and the basis functions ψ_k associated to both the new samples and the original samples on the boundary of e_l are recomputed to ensure that locality is preserved. In our refinement method, each basis function is written as a sum of linear B-splines:

$$\psi_k(x) = \sum_{i,j} \alpha^{i,j}_k \phi^j_i(x). \tag{6.6}$$

Then, our approximation can be expressed as:

$$P(x) = \sum_{i,j} \left(\sum_k \alpha^{i,j}_k p_k \right) \phi^j_i(x). \tag{6.7}$$

We can extend the definitions above to this alternative expression. We extend the definition of support to ϕ^j_i, where $S(\phi^j_i)$ is the region where it assumes non-zero values. Since locality was defined as $S(\psi_k) \subset N(x_k), \forall k$, an approximation preserves locality if and only if $S(\phi^j_i) \subset \mathcal{N}(x_k), \forall k$ such that $a^{i,j}_k \neq 0$. We can therefore make the following definition and remark:

Definition 5 (Local Point). *We say that a point $y^i_j \in \mathcal{A}$ is a local point of a B-spline ϕ^j_i if $S(\phi^j_i) \subset \mathcal{N}(y^i_j)$.*

Definition 6 (Local Sample Set)**.** *We define the local sample set* $\mathcal{L}_j^i$ *of a B-spline* ϕ_i^j *as the set of samples* x_k *that are local points of* ϕ_i^j*, i.e, such that* $S(\phi_i^j) \subset \mathcal{N}(x_k)$*.*

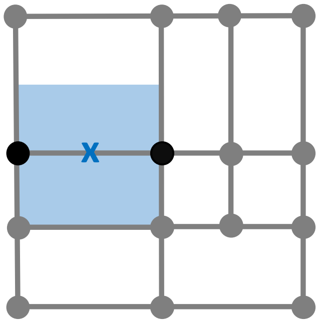


Figure 6-5: A linear B-spline ϕ_i^j is illustrated in blue: the blue "‘x"’ is the center and the region where it assumes non-zero values is shaded in light blue. The local sample set of this linear B-spline $\mathcal{L}(\phi_i^j)$, is defined as the set of samples x_k such that $S(\phi_i^j) \subset \mathcal{N}(x_k)$ and is illustrated in black.

Remark 1 (Locality)**.** *An approximation preserves locality if* $\alpha_k^{i,j} = 0, \ \forall k \notin \mathcal{L}_j^i$*.*

These defintions are useful when defining the algorithm that ensures locality and to prove that our algorithm also ensures *linear precision* (linear functions are recovered exactly) and the *partition of unity* property (the sum of all basis functions is one).

6.5 Adaptive Refinement Strategy

We discuss our method for refining elements in a K-d tree, which guarantees both locality and continuity.

6.5.1 Motivation

As discussed in previous work, an approximation over a discretized domain can be described from two perspectives: *elements* and *basis functions*. From the element perspective, the approximation is defined over each element by an interpolation of the samples on the boundary. From the basis function perspective, the approximation is described as a sum of basis functions weighted by sampled values.

Though element representations drive refinement operations that ensure locality, they introduce discontinuities along T-junctions on high-dimensional adaptive grids. Consider for example the standard scheme, where multi-linear interpolation is done on each element. Figure 6-6 illustrates the discontinuities at the boundary between elements at different levels that results from this approach and are not present in our method.

Refinement of basis functions, on the other hand, can be used to add detail while preserving continuity. Two approaches of basis refinement have been discussed in previous work:

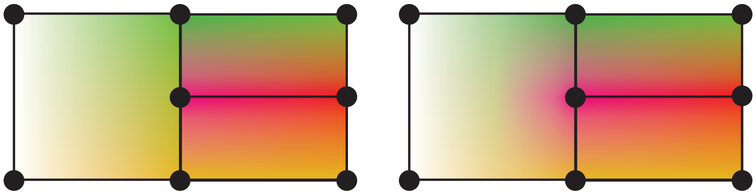


Figure 6-6: Comparison between multi-linear interpolation (left), where discontinuities appear along T-junctions, and our method (right), where the interpolation is continuous.

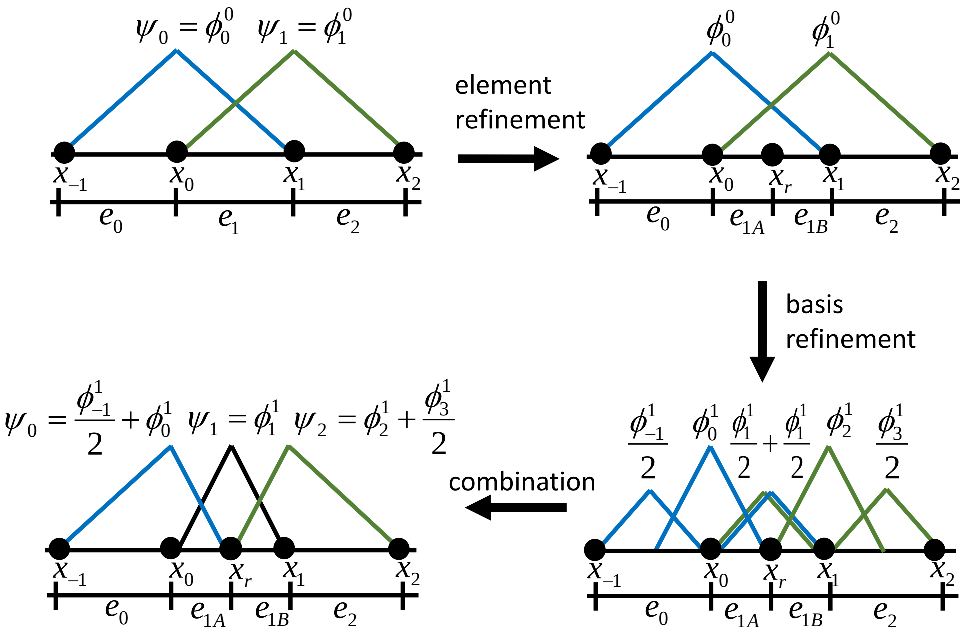

$\psi_0 = \phi_0^0$ $\psi_1 = \phi_1^0$
x_{-1} x_0 x_1 x_2
e_0 e_1 e_2
element refinement
ϕ_0^0 ϕ_1^0
x_{-1} x_0 x_r x_1 x_2
e_0 e_{1A} e_{1B} e_2
basis refinement
$\frac{\phi_{-1}^1}{2}$ ϕ_0^1 $\frac{\phi_1^1}{2} + \frac{\phi_1^1}{2}$ ϕ_2^1 $\frac{\phi_3^1}{2}$
x_{-1} x_0 x_r x_1 x_2
e_0 e_{1A} e_{1B} e_2
combination
$\psi_0 = \frac{\phi_{-1}^1}{2} + \phi_0^1$ $\psi_1 = \phi_1^1$ $\psi_2 = \phi_2^1 + \frac{\phi_3^1}{2}$
x_{-1} x_0 x_r x_1 x_2
e_0 e_{1A} e_{1B} e_2

Figure 6-7: Our method of combining element refinement with basis refinement: interpolated values inside each element only depend on the samples that lie on the boundary of that element.

hierarchical refinement, in which dilated basis functions are added in the center of an element Forsey and Bartels [1988]; and quasi-hierarchical refinement, which takes advantage of refinement relations and has the advantage of restricting the disparity between levels Grinspun et al. [2002]. Both of these basis refinement methods, however, have the disadvantage of not preserving locality (see Figure 6-8).

We propose an alternative method that combines the element and basis functions perspectives. Our algorithm does refinement based on elements to guarantee locality *but* uses basis functions to guarantee continuity. Drawing ideas from the quasi-hierarchical scheme for basis refinement, our algorithm takes advantage of refinement relations. Figure 6-7 illustrates our algorithm on a simple one-dimensional example. Initially, the approximation on

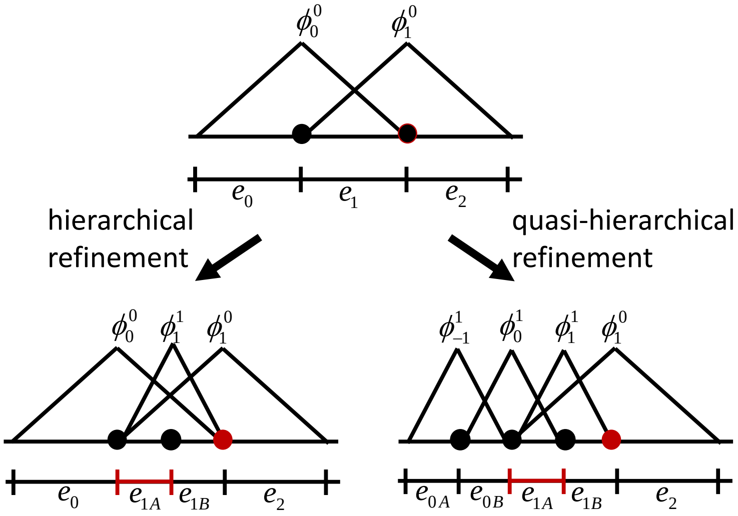

ϕ_0^0
ϕ_1^0
e_0
e_1
e_2
hierarchical refinement
quasi-hierarchical refinement
ϕ_0^0
ϕ_1^1
ϕ_1^0
e_0
e_{1A}
e_{1B}
e_2
ϕ_{-1}^1
ϕ_0^1
ϕ_1^1
ϕ_1^0
e_{0A}
e_{0B}
e_{1A}
e_{1B}
e_2

Figure 6-8: Hierarchical and quasi-hierarchical basis refinement strategies. Both of these schemes violate locality on element e_{1A} since the basis function ϕ_1^0 does not vanish on this element. The sample outside $\mathcal{B}(e_{1A})$ affecting this element is highlighted in red.

element e_1 is given by:

$$P^1(x) = p_0^1 \underbrace{\phi_0^0(x)}_{\psi_0(x)} + p_1^1 \underbrace{\phi_1^0(x)}_{\psi_1(x)}.$$

In the first step of our method, we refine the element e_1 by adding a sample x_r in center of e_1 and splitting e_1 into two new elements: e_{1A} and e_{1B}. In the second step, we refine all basis functions that overlap e_1; we can therefore express the approximation as

$$\begin{aligned} P^1(x) = p_0^1 & \underbrace{\left(\frac{1}{2}\phi_{-1}^1(x) + \phi_0^1(x) + \frac{1}{2}\phi_1^1(x)\right)}_{\psi_0(x)} \\ + p_1^1 & \underbrace{\left(\frac{1}{2}\phi_1^1(x) + \phi_2^1(x) + \frac{1}{2}\phi_3^1(x)\right)}_{\psi_1(x)}. \end{aligned}$$

Finally, we define the new basis function by associating the B-splines that break locality—in this case, ϕ_1^1—to the sample x_r. The resulting approximation is

$$\begin{aligned} P^1(x) = p_0^1 & \underbrace{\left(\frac{1}{2}\phi_{-1}^1(x) + \phi_0^1(x) +\right)}_{\psi_0(x)} \\ + p_1^1 & \underbrace{\left(\phi_2^1(x) + \frac{1}{2}\phi_3^1(x)\right)}_{\psi_1(x)} + p_r^1 \underbrace{\left(\phi_1^1(x)\right)}_{\psi_r(x)}. \end{aligned}$$

From this, we can write $P^{1A}(x) = p_0^{1A}\psi_0(x) + p_r^{1A}\psi_r(x)$ and $P^{1B}(x) = p_1^{1B}\psi_1(x) + p_r^{1B}\psi_r(x)$, guaranteeing locality on the new elements. Since our refinement strategy only involves refining B-spline functions using the refinement relations and regrouping them into composed basis functions, partition of unity is preserved. Moreover, we observe from the two equations above that, if $p_r^1 = (p_0^1 + p_1^1)/2$, then the refinement operation does not alter the solution, which indicates that linear precision is preserved. From Figure 6-7 we observe that this strategy is equivalent to standard element refinement in the one-dimensional case, in which the hanging nodes issue does not arise. However, this strategy can be generalized to high dimensions, allowing continuous interpolation on adaptive grids that preserve locality. Furthermore, this method can be extended to higher-order basis functions, such as cubic B-splines that allow smooth interpolations (Section 6.5.3).

6.5.2 Algorithm

The general algorithm in high dimensions for refining an element e_l given a split direction d is described by the following steps:

step 1: Refine the element e_l.
step 2: Refine the basis functions overlapping e_l.
step 3: Redefine ψ_k to ensure locality.

Definition 7 (Common Cuboid)**.** *Given any set of elements e_l with non-empty intersection, we define the common cuboid as the region $R = \cap e_l$. Since each element is a K-dimensional cuboid, R is a (possibly lower dimensional) cuboid.*

In step 1, the element e_l is refined, creating two child elements e_{lA} and e_{lB} and new samples are added to guarantee that there exists a sample on the corners of every common cuboid (see Figure 6-9). Only intersections of e_{lA} and e_{lB} need to be checked and samples are added only if they do not previously exist and are generated by evaluating the CAD shape.

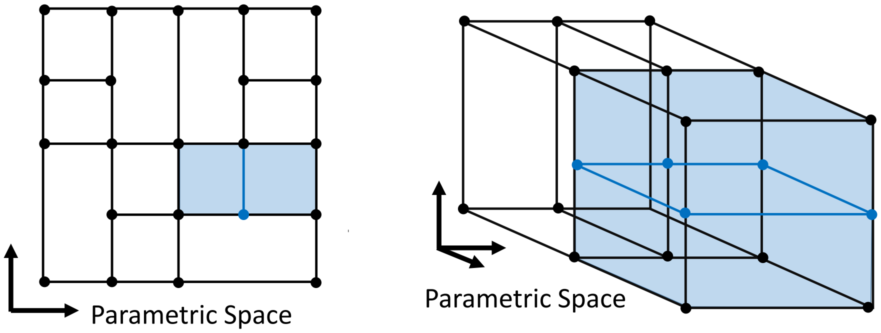


Figure 6-9: Example of samples added for a given split in 2D and 3D. Split element, split plane and added samples are shown in blue.

In step 2, we use the refinement relations to refine all the B-splines ϕ_i^j overlapping e_l where the size of their support $S(\phi_i^j)$ is larger than the size of the element e_l in direction d. Since locality is preserved in every iteration, only the B-splines ϕ_i^j in the terms of $\psi_k, x_k \in \mathcal{B}(e_l)$

(Equation 6.6) can overlap e_l and therefore need to be checked. We can show that at each iteration, step 2 needs to perform at most one level of refinement (see Section S2 of supplemental material). This step updates the values of $\alpha_k^{i,j}$. By substituting Equation 6.4 into Equation 6.6, the updated values, $\bar{\alpha}_k^{i,j}$, are:

$$\begin{cases} \bar{\alpha}_k^{n,j+1} = \alpha_k^{n,j+1} + a_{in}^{(j+1)} \alpha_k^{i,j}, \forall n, k \\ \bar{\alpha}_k^{i,j} = 0, \quad \forall k \end{cases} \tag{6.8}$$

From the properties of refinement relations, this does not alter the summed value in Equation 6.7 and, therefore, the properties of partition of unity and linear precision are preserved after this step.

Finally, in step 3, we redefine the basis functions ψ_k to enforce locality, which is done by updating the values $\alpha_k^{i,j}$ (see Equation 6.6). There are multiple assignments of these values that guarantee locality. From Remark 1, locality can be guaranteed simply by setting the coefficients that violate locality to zero ($\alpha_k^{i,j} = 0, \ \forall x_k \notin \mathcal{L}_i^j$). Simply zeroing out these coefficients, however, would make the resulting approximation function $P(x)$ break important interpolation properties. We therefore propose a refinement strategy that guarantees locality but also enforces a partition of unity and linear precision.

Enforcing a Partition of Unity From Equation 6.7, a partition of unity is guaranteed if the updated values of $\alpha_k^{i,j}$, $\bar{\alpha}_k^{i,j}$, satisfy the following property for every ϕ_i^j:

$$\sum_k \bar{\alpha}_k^{i,j} = \sum_k \alpha_k^{i,j} \tag{6.9}$$

Therefore, in order to ensure both locality and a partition of unity, we first select all of the B-splines ϕ_i^j that violate locality (according to Remark 1) with the refinement of e_l. Since we assume that locality was preserved in previous iterations, only the B-splines ϕ_i^j in the terms of $\psi_k, x_k \in \mathcal{B}(e_l)$ (Equation 6.6) can overlap e_l and therefore need to be checked. Then, for each of these B-splines, we must zero out the coefficients $\alpha_k^{i,j}$, $x_k \notin \mathcal{L}_i^j$ and distribute their added value amongst other coefficients $\alpha_k^{i,j}$, $x_k \in \mathcal{L}_i^j$ to enforce that the sum above is preserved.

To achieve this, we must show that the set $\mathcal{L}_i^j$ is not empty and define a method for finding samples in this set and using them to update $\alpha_k^{i,j}$.

Local Point Lemma. *For every ϕ_i^j there exists a local point y_i^j.*

The proof is given in the supplemental material.

Given a local point y_i^j, we propose the following *reallocation procedure*. Let R_i^j be the common cuboid defined by the intersection of the elements $e_l \in \mathcal{N}(y_i^j)$. In Figure 6-5, R_i^j is the line segment between the two black samples. The samples x_r on the corners of this cuboid are guaranteed to exist by step 1.

Claim. *The corner samples $x_r \in R_i^j$ are in $\mathcal{L}_i^j$.*

Proof. Let $e_l \in \mathcal{N}(y_i^j)$. Then $x_r \in \cap e_l$ implies $x_r \in \mathcal{B}(e_l)$ and therefore $e_l \in \mathcal{N}(x_r)$. From this we conclude that $\mathcal{N}(y_i^j) \subset \mathcal{N}(x_r)$, $\forall x_r \in R_i^j$. Since we assume $S(\phi_i^j) \subset \mathcal{N}(y_i^j)$, from the definition of $\mathcal{L}$ it follows that $x_r \in \mathcal{L}_i^j$, $\forall x_r \in R_i^j$. □

Since $y_i^j \in R_i^j$, we can express it as the convex combination $y_i^j = \sum \beta_r^{i,j} x_r, \sum \beta_r^{i,j} = 1$ using multi-linear weights $\beta^{i,j}r$ where x_r are the samples in R_i^j. Let $\alpha_i^j = \sum_k \alpha_k^{i,j}$. We then set

$$\bar{\alpha}_r^{i,j} = \begin{cases} \alpha_i^j \beta_r^{i,j}, & x_r \in R_i^j \\ 0, & \text{otherwise.} \end{cases} \tag{6.10}$$

Partition of unity is preserved since $\sum_r \beta^{i,j} = 1$. This completes the realocation procedure.

It can be shown that if c_i^j is the center of the B-splines ϕ_i^j, then c_i^j is a local point of ϕ_i^j. We can therefore use the realocation procedure with $y_i^j = c_i^j$ to find a solution that guarantees locality and a partition of unity.

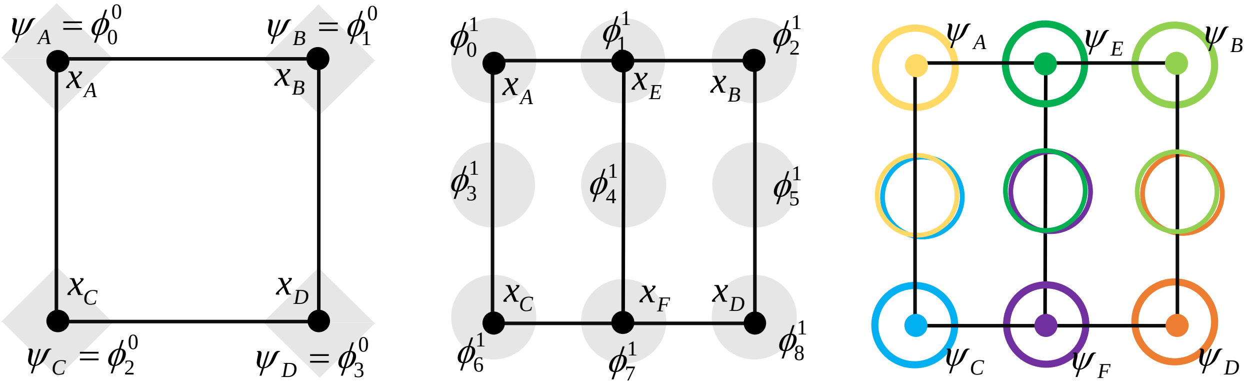


Figure 6-10: Illustration of refinement in two dimensions. Letters are used to index x_k for clarity. When the element is split, new samples are added, the B-splines ϕ_i^j are refined and the new basis functions ψ_i^j are recomputed to ensure locality.

A 2D example of this is shown in Figure 6-10. In the first step, we split the element generating samples at x_E and x_F. In the second step, we refine all the basis function shown since they all overlap the element. In the third step we find the B-splines that violate locality (according to Remark 1). For example, the B-spline ϕ_4^1 violates locality since $\alpha_A^{4,1} = \alpha_B^{4,1} = \alpha_C^{4,1} = \alpha_D^{4,1} = 1/4$ and the samples x_A, x_B, x_C and X_D are not in $\mathcal{L}_4^1$. As discussed above, we compute R_4^1,which is the line segment between x_E and x_F. We express c_4^1 as a function of samples at R_4^1, $c_4^1 = (x_E + x_F)/2$. We therefore we update the values of α setting $\bar{\alpha}_E^{4,1} = \bar{\alpha}_F^{4,1} = 1/2$ and zero elsewhere. Similarly, we update the coefficients of ϕ_1^1 and ϕ_7^1, which violates locality. From this we can redefine $\psi_E = \phi_1^1 + \phi_4^1/2$, $\psi_F = \phi_7^1 + \phi_4^1/2$, $\psi_A = \phi_0^1 + \phi_3^1/2$, $\psi_B = \phi_2^1 + \phi_5^1/2$, $\psi_C = \phi_0^6 + \phi_3^1/2$, and $\psi_D = \phi_8^1 + \phi_5^1/2$. The colors on the right in Figure 6-10 indicate the influence of each sample, which are local after this redefinition.

Enforcing Linear Precision In addition to locality and partition of unity, we would like our interpolation to exactly reproduce linear functions. We will propose a method for defining the $\bar{\alpha}_k^{i,j}$ such that if the evaluations function $x_k \longmapsto p_k$ is linear and linear precision was enforced in all previous iterations, then

$$\sum_k \bar{\alpha}_k^{i,j} p_k = \sum_k \alpha_k^{i,j} p_k \tag{6.11}$$

From Equation 6.7, this implies that, in the linear case, the approximation does not change with refinement and therefore continues to exactly reproduces linear functions after each iteration.

For intuition, let us first consider the special case when $\alpha_i^j = \sum_k \alpha_k^{i,j} = 1$. Under this assumption, the realocation procedure with $y_i^j = c_i^j$ results in $c_i^j = \sum_k \bar{\alpha}_k^{i,j} x_k$, since $\bar{\alpha}_r^{i,j} = \beta_r^{i,j}$, $\forall x_r \in R_i^j$. Let $p_{c_i^j}$ be the evaluation at c_i^j. If the evaluation function is linear, then above expression yields $p_{c_i^j} = \sum_k \bar{\alpha}_k^{i,j} p_k$. On the other hand, If linear precision was guaranteed in all previous iterations, then $p_{c_i^j} = P(c_i^j)$. Since every B-spline evaluates to 1 on its center and partition of unity is guaranteed, Equation 6.7 yields $P(c_i^j) = \sum_k \alpha_k^{i,j} p_k$. From this we conclude that Equation 6.11 holds and therefore linear precision is preserved.

Unfortunately, in general, we cannot guarantee that $\alpha_i^j = 1$. While this was true in the example in Figure 6-10, this is usually not the case in high dimensions and after multiple iterations. Our solution therefore is to use the realocation procedure with $y_i^j = \sum_k \alpha_k^{i,j} x_k / \alpha_i^j$ (notice that when $\sum_k \alpha_k^{i,j} = 1$, $y_i^j = c_i^j$). This is possible because we can prove that for this definition of y_i^j, $S(\phi_i^j) \subset \mathcal{N}(y_i^j)$ (supplemental material). By directly following the sequence of arguments above, we can show that, for this definition of y_i^j, linear precision is guaranteed in the general case.

6.5.3 Extension to Cubic B-splines

To extend our method to cubic B-splines, we first redefine locality. On a uniform grid, cubic B-splines generate smooth interpolations by introducing dependencies on adjacent samples. We therefore define the neighborhood $\overline{\mathcal{N}}(x_k)$ of a sample x_k as the set of elements that either contain the sample x_k or are adjacent to some element that contains the sample x_k. Analogously, we define the neighborhood $\overline{\mathcal{B}}(e_l)$ of an element e_l as the set of samples on its boundary and on the boundaries of its adjacent elements (see Figure 6-11).

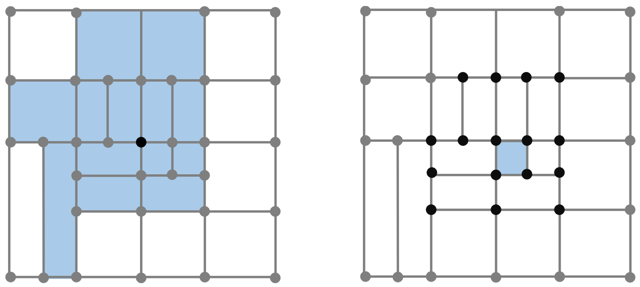


Figure 6-11: The neighborhood $\overline{\mathcal{N}}(x_k)$ of a sample x_k (left) and the neighborhood $\overline{\mathcal{B}}(e_l)$ of an element e_l (right) for cubic B-splines.

As in the linear case, we say that an approximation preserves locality if $\forall k, S(\psi_k) \in \overline{N}(x_k)$; or, equivalently, if, for each element e_l, any sample x_k whose associated basis function ψ_k does not vanish on e_l is contained in the neighborhood $\overline{\mathcal{B}}(e_l)$ of that element. We use the same method described in Algorithm 1, refining cubic B-splies in step 2 if their support is larger than twice the size of the refined element e_l in direction d. We use following refinement relation for cubic B-splines (see Figure 6-12):

ALGORITHM 1: Refine (e_l) along direction d

```
// Step 1 (Element Refinement)
create child nodes $e_{iA}$ and $e_{iB}$;
create new samples on corners of every common cuboid;
// Step 2 (Basis Refinement)
forall $\phi_i^j$ which overlaps $e_l$ do
    if $S(\phi_i^j(x))$ larger than $e_l$ along direction $d$ then
        // refine $\phi_i^j$
        forall $k, x_k \in \mathcal{L}_i^j$ do
            $\bar{\alpha}_k^{n,j+1} = \alpha_k^{n,j+1} + a_{in}^{(j+1)} \alpha_k^{i,j}, \forall n$;
            $\bar{\alpha}_k^{i,j} = 0$;
        end
    end
end
// Step 3 (Combination)
forall $\phi_i^j$ which overlaps $e_l$ do
    if $\phi_i^j$ violates locality then
        set $y_i^j = \sum_k \alpha_k^{i,j} x_k / (\sum_k \alpha_k^{i,j})$;
        set $R_i^j = \cap e_l, \forall e_l \in \mathcal{N}(y_i^j)$;
        compute $\beta_r^{i,j}$ such that $y_i^j = \sum \beta_r^{i,j} x_r$, $x_r \in R_i^j$;
        compute the updated values of $\alpha$, $\bar{\alpha}$:
            $\bar{\alpha}_r^{i,j} = \left(\sum_k \alpha_k^{i,j}\right) \beta_r^{i,j}$, $x_r \in R_i^j$
            $\bar{\alpha}_r^{i,j} = 0$, otherwise
    end
end
```

$$\begin{aligned}\phi_i^{(j)}(x) = \; & \tfrac{1}{8}\phi_{(2i-2)}^{(j+1)}(x) + \tfrac{1}{2}\phi_{(2i-1)}^{(j+1)}(x) + \tfrac{6}{8}\phi_{(2i)}^{(j+1)}(x) \\ & + \tfrac{1}{2}\phi_{(2i+1)}^{(j+1)}(x) + \tfrac{1}{8}\phi_{(2i+2)}^{(j+1)}(x).\end{aligned} \tag{6.12}$$

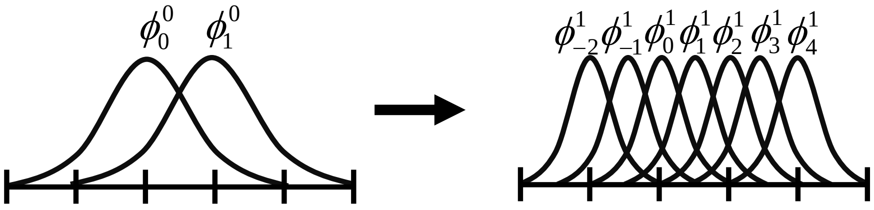


Figure 6-12: Refinement of cubic B-splines.

To compute the approximation $P^l(x)$ at $x \in e_l$, we need the meshes from all samples $x_k \in \overline{\mathcal{B}}(e_l)$ to be represented in the canonical format of e_l. If $x_k \in \overline{\mathcal{B}}(e_l) \setminus \mathcal{B}(e_l)$, then

$x_k \in \mathcal{B}(e_{\bar{l}})$, where $e_{\bar{l}}$ is adjacent to e_l. Let $x_{\bar{k}}$ be a sample in $\mathcal{B}(e_{\bar{l}}) \cap \mathcal{B}(e_l)$ (see Figure 6-13). Since $p^l_{\bar{k}}$ and $p^{\bar{l}}_{\bar{k}}$ are stored during our pre-computational phase and have the same geometry, we can compute a mapping $F : p^{\bar{l}}_{\bar{k}} \to p^l_{\bar{k}}$ using barycentric coordinates and then apply it to $p^{\bar{l}}_k$ to obtain p^l_k. Locality ensures that this mapping only has to be done once for each sample guaranteeing that errors do not accumulate.

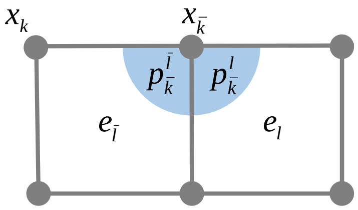


Figure 6-13: The mapping $F : p^{\bar{l}}_{\bar{k}} \to p^l_{\bar{k}}$ is used to obtain p^l_k for $x_k \in \overline{\mathcal{B}}(e_l) \setminus \mathcal{B}(e_l)$.

6.6 Homeomorphic Mapping

In this section we discuss how we define a homeomorphic map that allows for representing a mesh p_A in the canonical format of a given element, p^l. The resulting mesh p^l_A should have the same geometry as p_A and the same combinatorics as the canonical reference p_l (see Figure 6-17). To this end, we need to establish a dense correspondence between two meshes. Surface correspondence on meshes has been extensively studied in previous work Van Kaick et al. [2011]. Our problem is distinct because we can take advantage of the CAD referencing methods to establish partial correspondence. As we will show in this section, this correspondence is not necessarily complete since the topology of the internal CAD representations (B-rep) may differ even if the geometry varies smoothly. We therefore propose an algorithm that combines CAD data analysis with mesh surface correspondence algorithms from previous work.

6.6.1 Motivation

As previously discussed, CAD systems use B-reps to represent solid models, which are generated by computing a list of features. A referencing scheme is used to determine the parts of the model to which features are applied, allowing these to be correctly re-generated if the model is modified. It is very common that parameter changes affect the topology of the B-rep, even in cases in which the geometry varies smoothly. Because references depend on the feature history, not the B-rep itself, they are robust to topological changes that do not affect the referenced element. An example of this is shown in Figure 6-14.

The CAD system's API can be used to index each topological entity of the B-rep (faces, edges and vertices) from their internal referencing scheme. However, because faces can merge or split, we cannot generate a mapping simply from the correspondence between each face. In order to ensure that our method is robust to topological changes on the B-rep, we propose using the information from the CAD system to establish sparse correspondences on the surface of the shapes. This is then used to establish a dense correspondence map on the

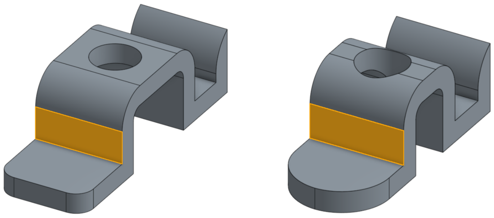


Figure 6-14: Example of referencing in Onshape. The user applies a feature to a given face which is highlighted (left), and then changes parameter making other faces and edges of the model merge or split (right). Onshape's referencing scheme guarantees that the feature will still be applied to the correct face even after these changes are made.

shape boundaries. Following the approach of previous work ([Aigerman et al., 2014, 2015b, Kraevoy and Sheffer, 2004, Lee et al., 1999, Praun et al., 2001, Schreiner et al., 2004] to name a few) we use this sparse correspondence to split both source and target meshes into a common patch layout and compute the parameterization for each patch. Finally, we compute the volumetric mesh, p^l_A, by propagating the boundary distortion map to all tetrahedrons in the interior.

6.6.2 Algorithm

CAD Referencing When sampling a point in the parametric domain x_k, we use a CAD system's API to update the parameters x_k, evaluate the model (i.e., recompute the feature list) and export the mesh p_k. Further, we use the API to query for a set of control variables that we define on the surface of the model in order to establish correspondences. Our control variables are a set of points with identifiers (IDs) and a set of paths between points. These control variables are set individually for each parameter configuration. We establish correspondences only when control variables with the same ID exist in both source and target meshes, addressing the case of varying B-rep topologies.

To generate these control variables, we designed a feature that references all of the vertices and edges of the solid model and tags each of these entities with a unique index. We add this feature to the end of the feature history. The reason we cannot simply use this index as the ID for each vertex and edge is that, though some references will break when entities vanish, others will be preserved even if the entities undergo topological changes. The referencing in CAD systems is designed this way in order to conserve certain operations even when the entities to which they are applied merge or split (see Figure 6-15). Though this is very powerful for CAD design, it is problematic for our application, since using these indices directly would allow us to create correspondences between vertices that collapse to a point or between two edges that are adjacent in one model and edges that are not adjacent in the other. To ensure a smooth surface mapping, these correspondences must be avoided since they will lead to shrinking the triangles in between to zero.

To address this problem, we establish correspondences only between elements of the B-rep which undergo no topological changes. To this end, we define the ID for each vertex as the vertex index and a (sorted) vector of indices of adjacent edges. Analogously, we define the ID for each edge as the edge index and a (sorted) vector of indices of adjacent vertices. This simple scheme is sufficient to ensure that matching IDs exist only between elements

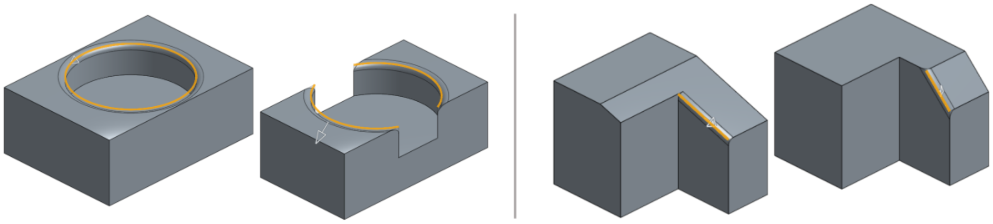


Figure 6-15: Two examples of a fillet feature applied to edges. After this feature is defined, parameter changes on earlier features in the feature list split the edges into two parts. The way the edge references are handled in order to re-generate the fillet depends on the feature history, not the geometry. On the model on the left, the CAD system applies the fillet to both edges generated by the split; on the model on the right, the fillet is applied to only one of the edges.

with matching topologies.

Once IDs are established for every vertex and edge, control points are then set by computing the position of each referenced vertex and sampling each referenced edge. Sampling on edges is easily performed by the geometric kernel using the API. Though we would like the number of samples per edge to be proportional to the edge length, we require similar sampling on source and target meshes in order to establish correspondences. Since edge sizes can vary significantly when parameters are changed, we sample each edge by hierarchical subdivision until the length of each segment is smaller than a given threshold, and generate IDs for each sample according to this subdivision. In addition to the control points, our API call also returns a list of paths that connect pairs of points that are consecutive along an edge.

Surface Mapping The common patch layout defines graphs on both source and target meshes. We use the set of control points with matching IDs to define the nodes on this graph. Edges on this graph define paths on the source and target meshes connecting these vertices. In the first stage, we use the paths extracted from the CAD system to create edges in this graph (see second row of Figure 6-16).

We then add edges to the patch layout, following the approach of previous work Kraevoy and Sheffer [2004]. To this end, we first compute a list of candidate edges by computing shortest paths between every pair of control points in both source and target meshes and sorting this candidate list by the sum of the lengths of the source and target paths. These paths are constrained to (a) not intersect each other, (b) maintain the cyclical order, and (c) not place corresponding control points in two different patches. We use the candidate list to incrementally add edges to the patch layout. Every time a new edge is added, the candidate list is updated to ensure the constraints are satisfied.

Edges are selected first to ensure the graph is connected and then to ensure that every patch is simple. Finally, more edges are added to further subdivide large patches for better parametrization. In this step, however, we add an edge only if the shortest path length is significantly smaller than the path that connects these two points on the patch layout. Adding this restriction is necessary for our application because details such as small fillets or holes can generate a large number of control points that are very close to each other. Finally,

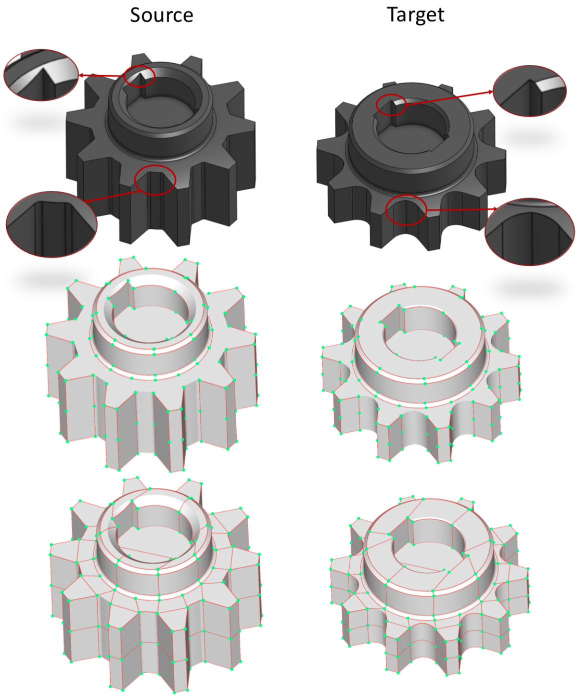


Figure 6-16: Common patch layout given a source and target geometry. Top to bottom: B-rep with highlighted topological changes, paths extracted from CAD, complete patch layout.

we refine paths to geodesic paths. Using geodesic paths smooths the boundary, making the paths on the target and source meshes more consistent. This is done only after the paths are selected because it is a time-consuming step. Finally, we construct a mapping between each path pair using a mean-value parametrization technique Floater [2003] to map each patch onto a disk and then composing the two maps to establish a mapping from the source to target patches.

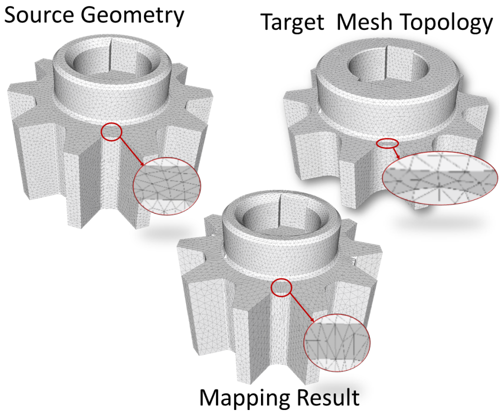


Figure 6-17: Mapping result. Given a source geometry and a target mesh topology the mapping outputs a mesh with the source geometry and target topology.

Interior mapping Since, in our problem, meshes are reasonably similar, we can map the surface mapping to the interior by solving a linear elastic FEM problem. We use the surface map as the Dirichlet boundary conditions for the FEM simulation, and compute the volumetric deformation for the interior map.

Boundary Conditions In addition to mapping the shapes, we also need to establish consistent boundary conditions for our simulations. Consider, for example, a chair model in which the force is applied to the seat, which can vary in length and width. In order to simulate this result, we need to know which mesh vertices correspond to the seat as the mesh varies. We address this problem by developing a boundary condition feature to be added at the end of a solid model's feature list. These features take as inputs the faces of the model and the information of the boundary condition type (forces, temperature, etc). Similar to the discussion above, we use the CAD referencing scheme to find the mesh regions that correspond to the selected faces. This allows us to establish consistent boundary regions during interpolation.

6.7 Results

While our method can be applied to any CAD system, we have implemented the discussed techniques using Onshape's API.

Our precomputed data and interpolation scheme allow visualization of geometry variations in real time. Our tool can further display to the user physical properties that depend on the geometry and can be computed at interactive rates. In our experiments, we evaluate mass, surface area, drag, and moment of inertia. Other physical properties that are defined over the mesh and require expensive computations can be precomputed for every sample and interpolated with our scheme. Different applications require analysis of different properties. When constructing the K-d tree, we define the error metric for determining if an element should be refined by either comparing the geometry (with Hausdorff distances) or the error on the precomputed simulation in applications where these properties are added.

Figure 4-1 and the first four rows of Figure 6-18 show examples with precomputed stress analysis. Figure 4-1 shows a wrench that is constrained to be fixed in the faces which are in contact with a bolt head and has a force pulling down on the handle. The result displays the moment of inertia, mass and stress distribution as three parameters vary: the head radius, the handle length and the fillet connecting this two parts.

The first row of Figure 6-18 shows a rocker arm on a bike that is fixed to the suspension component and has a force from the rear wheel. The four parameters include directions and length of the inner holes, thickness of the part and fillet radius on the border. Typically, an engineer will try to minimize the weight and drag of a model like this while keeping stress below a certain threshold. Our interface displays all properties in real time. This model highlights an advantage of this visualization tool over a direct optimization since both the thickness and the hole size affect all variables and this tool allows users to choose from the multiple configurations that meet the design criteria.

The second row shows a chair with precomputed stress analysis which results from a force on the seat. Our mapping of boundary conditions allows the force to be uniformly applied

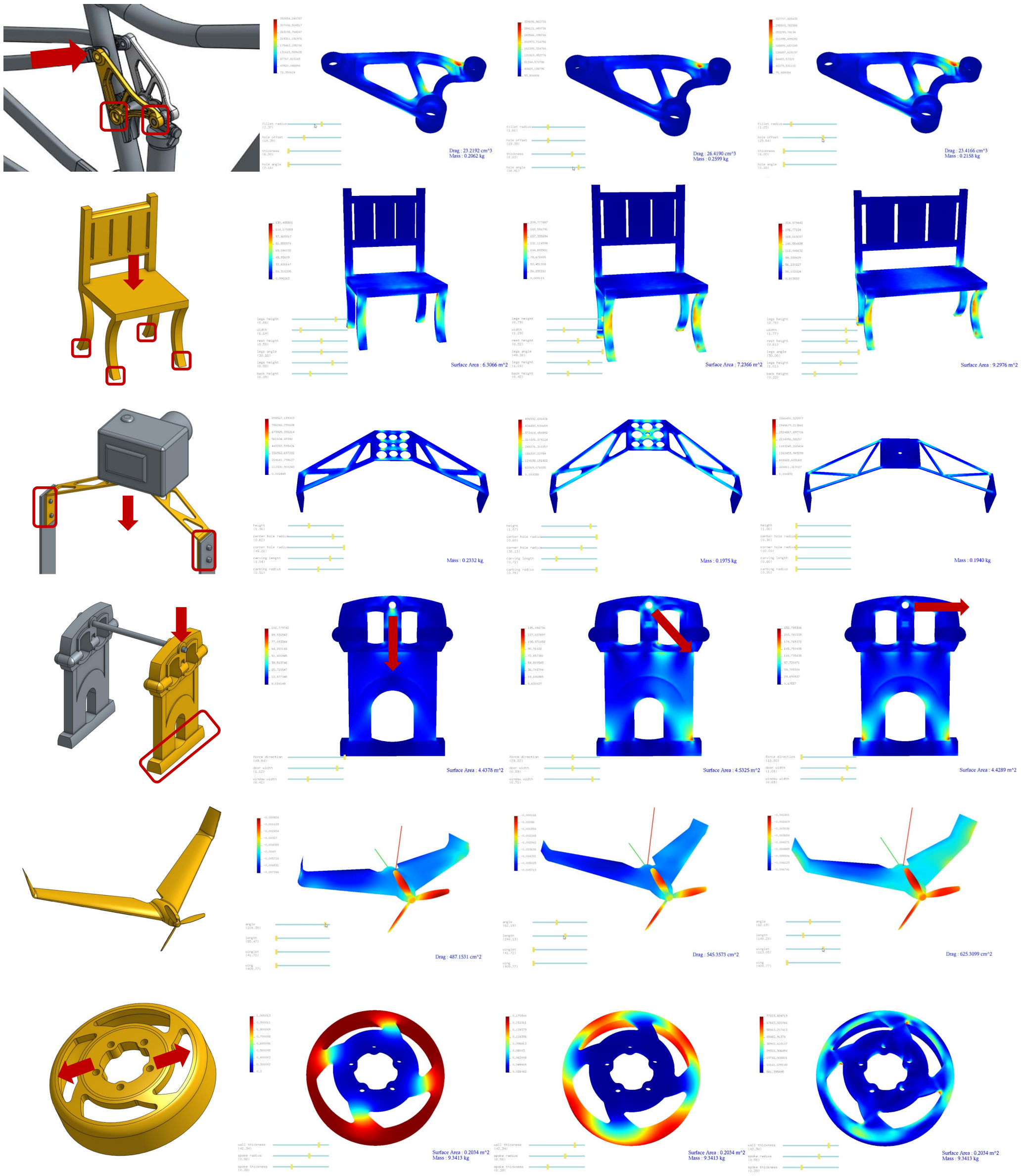


Figure 6-18: Examples of interactive visualization. On the left we show each model in red boxes and the regions with fixed boundary conditions and forces with red arrows. On the right we show results from our visualization interface. As the user varying the parameters, geometry is updated is real time. The top four rows show pre-computed stress analysis. On the forth row the force direction also varies and is illustrated with a red arrow. The fifth row shows a result with fluid simulation. The last row shows a results with thermoelastic simulation where the colors display (from left to right) heat distribution, deformation and stress.

on the surface of the seat across parameter variations which affect its dimensions. In this example a total of six parameters can be exposed since a chair does not need to interface with other models in an assembly, allowing greater variability.

The third row is a camera mount, which has fixed sizes and a downwards force from the weight of the camera. The engineer aimed at minimizing both the mass and the deformation while making the stress distribution as even as possible. Using our tool to view all these properties at interactive rates, the engineer can find an optimal solution without needing to come up with reasonable weights for a composed objective function to be used in optimization. The holes were designed as circular patches with varying radii which can be fabricated with punching, which is far less expensive than milling, illustrating how CAD data can be applied for optimization with fabrication constraints.

The fourth row is a swing castle where the dimensions of the door and windows are parametrized. Since this model requires expensive coating, surface area is displayed along with the pre-computed stress analysis. In this example, the direction of the force on the bar varies as the child swings. This direction is treated as a third parameter in our interface allowing users to visualize the stress distribution as both the geometry and the force vary.

The fifth row shows an example of a toy airplane for which the air pressure distribution was precomputed using a compressible flow solver. Our interactive tool displays this property together with moment of inertia and drag from a frontal wind which are directly computed from the geometry at interactive rates.

Finally, the sixth row shows an example where thermal expansion is coupled with the stress from the brake hub. The heat source is uniform on the inner walls while the forces from the brake are directional. We measure the stress that results from both the directional forces and the heat deformations. We store the pre-computed heat distribution, stress, and deformation. The engineer can explore the shape and size of the inner holes to normalize the stress distribution and also vary the thickness of the rim. We observe that as the latter increases, the deformation on the rim becomes more uniform since the impact of the heat outweighs the directional forces.

Table 6.1 describes the number of parameters, levels of the K-d tree, total sampled nodes, average mesh size for each model, and overall number of meshes stored. For precomputation, the time to evaluate each instance (specific parameter values) is the latency of the CAD evaluation and physical simulation. In the examples, geometry evaluation for each instance took about one minute, including B-rep regeneration from Onshape and tetrahedralization. Simulation times range from 5 to 10 minutes for stress analysis, and 2 to 3 minutes for thermoelastic and air pressure distribution. Our interactive viewer updates all examples in real time.

Table 6.1: The number of parameters (K), levels of the K-d tree, total sampled nodes, average mesh size (in number of tetrahedrons), and total storage (in number of stored meshes) for each model.

model	K	levels	samples	mesh size	storage
wrench	3	5	49	386k	112
chair	6	7	777	61k	4224
bike frame	4	6	103	288k	304
camera mount	5	6	259	291k	1056
swing	3	5	58	211k	144
plane	4	5	148	429k	432
drum brake	3	6	72	356k	208

Table 6.2 shows the approximation error for both geometry and elastic FEM simulation for each model. This is measured by sampling 10 random points in the domain and comparing the evaluated shapes and physical properties with our approximation. We observe that while geometry reconstruction is quite accurate, physical properties do not interpolate as well since they are highly non-linear. Still, we argue that it is accurate enough for a visualization cycle of gaining intuition since it clearly highlights how each parameter affects the physical properties of the model and which ones are the most relevant. When high accuracy is crucial, designers can use this tool to select optimal configurations and then run simulations at a second stage for a small number of selected parameters. We argue, however, that in most practical scenarios this second stage is not necessary because the approximation error is reasonable given engineering safety factors.

Table 6.2: Relative approximation error on geometry and elastic FEM for all example models.

model	geometry max	geometry 99%	FEM max	FEM 99%
wrench	0.0028	0.0004	0.2138	0.0393
chair	0.0243	0.0022	0.3756	0.1130
bike frame	0.0112	0.0003	0.1403	0.0281
camera mount	0.0009	0.0002	0.4107	0.1861
swing	0.0013	0.0002	0.4750	0.2042
plane	0.0162	0.0055	-	-
drum brake	0.0016	0.0006	0.3095	0.0490

When engineers design a part, they typically use simulation at multiple configurations in order to optimize physical properties. Since each simulation takes a long time, they use their training and expertize to pick certain configurations for testing and then to tweak the parameters based on these results. This process requires expert knowledge because each simulation takes a very long time and the more experience they have the fewer iterations are required Foshey et al. [2017]. For these reasons, interactive tools such as this one have the potential to not only facilitate the design process for engineers but also lower the barrier for novice users. In addition, it can be used to help less experienced engineers gain this intuition since it is a very informative way of illustrating how geometry impacts physical properties.

6.7.1 Application in Shape Optimization

Our method can also be used for automatic shape optimization. For this application we use cubic B-splines, which guarantee smooth interpolations. Using Equation 6.1, we can define the derivatives of our approximation as

$$P'(x) = \sum_k p_k \psi'_k(x), \tag{6.13}$$

which can be expressed analytically since $\psi_k(x)$ is a sum of weighted B-splines. We use this to drive a gradient-based algorithm for optimization over objective functions defined on the mesh. We use an interior point method and IpOpt for implementation.

Figure 6-19 shows an example of an optimization result where the objective function is defined as the integral of the stress over the volume. This arbor press has five parameters, two that define the dimensions and three that are structural (fillet radius, side thickness and hole size). Since different applications require different dimensions, companies will typically sell a range of similar shapes with varying heights and widths. This shows a typical use case for our method when a shape can be pre-computed once and then optimized for different use cases. The figure shows results for two configurations where heights and widths are fixed and the structural variables are optimized.

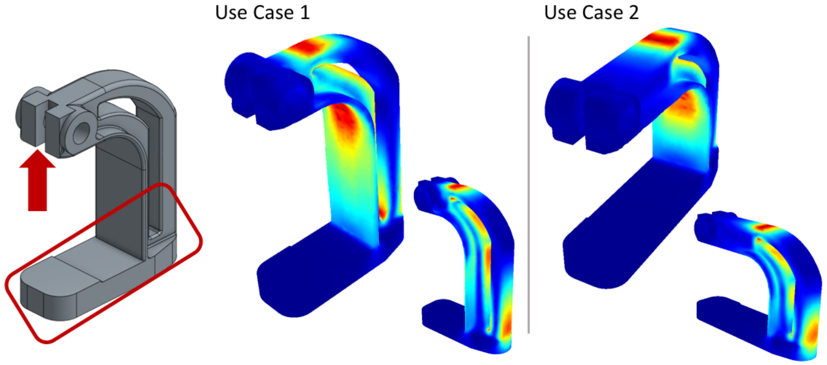


Figure 6-19: Result of shape optimization. Stress is minimized on an arbor press for two different use cases, a tall one is shown on the right and a short one on the left.

6.7.2 Limitations

While the main technical contributions of this approach is the novel interpolation algorithm, this cannot be applied for discrete parameters or topological changes. In such scenarios, our method can still be used to interpolate the continuous variations between discrete changes of parameters or topology. In the future it would be interesting to explore this technique in tools that combine discrete and continuous exploration such as the one proposed by Bao at el. [2013].

Another limitation of our precomputation scheme is that it does not scale well with the number of parameters. For CAD applications this is not a problem because the number of parameters that can be exposed tend to be small due to large number of constraints driven by manufacturing considerations or the need to interface with other models [Baran, 2017, Shugrina et al., 2015].

Storing 2^K meshes for every sample is arguably another limitation of our approach. In our implementation, this was not a problem even in the case of high resolution meshes. In the future, however, it could be interesting to incorporate compression schemes on the meshes Alliez and Gotsman [2005]. Additionally the physical properties computed can also be compressed, as is done in this work for corner vibration modes Langlois et al. [2014].

Since the interface with the CAD system allows detailed control variables that highlight the sharp features of the model and since shape variations from parameter changes are relatively small, we were able to extract satisfactory results with simple surface correspondence algorithms. In the future, it would be interesting to combine our method for extracting control variables from CAD with more recent approaches such as Aigerman et al. [2015a], which would guarantee more smoothness on patch boarders. It would also be interesting to experiment with mesh morphing algorithms that are not linear Alexa et al. [2000].

6.8 Discussion

In this chapter we propose a precomputation approach that allows interactive exploration of a design space with real-time performance feedback. Our main technical contributions is an interpolation scheme for adaptive grids that simultaneously guarantees continuity/smoothness and locality. This is a general interpolation algorithm that we hope will find many applications

We show examples of how our method can be coupled with different physics simulations to allow designers to interactively or automatically optimize shapes for different objectives. Since shape regeneration from CAD parameters and accurate evaluations of their physical properties are inherently slow, the existing workflow for optimizing parameters is not only time consuming, but also requires expertise since only a small set of variations can be tested in a reasonable amount of time. In this context, our tool not only helps experts to efficiently search the design space, but also helps bring down the design barrier to casual users.

The main limitation of this approach is that by sampling in the design space it suffers from the curse of dimensionality. While CAD design parameters are typically low dimensional, this ends up being an important limitation to applying this technique to explore more complex design spaces. In the next chapter, we argue that in designing for functionality it is not necessary to represent the full design space, since only a subset of solutions correspond to optimal design trade-offs. By representing only this subset, which lies in a much lower dimensional *performance* space, our approach not only scales to large design spaces, but also allows for a more meaningful exploration based on performance trade-offs.

Chapter 7

Interactive Performance-Space Exploration

7.1 Introduction

The previous chapter discusses a technique for exploring the *design* space with real-time feedback on *performance.* This chapter will be dedicated to solving the inverse problem: given a set of *performance* objectives, find the optimal *design.* This can be posed as an optimization problem over the design space for a given performance objective. As previously discussed, the fundamental challenge with such optimization techniques is that design for manufacturing typically involves multiple *conflicting* objectives.

Since it is impossible to optimize more than one criterion at a time, standard optimization approaches require expressing a set of performance criteria in a combined objective function that balances incompatible features. The typical approach is to use weighted combinations of different performance metrics. For example, in the case where there are two performance metrics, an objective could be expressed as $\alpha f_1(x) + (1-\alpha) f_2(x)$, where the f_i's are performance functions defined on the design space and $0 \leq \alpha \leq 1$ is a proxy for the trade-off between f_1 and f_2.

Unfortunately, such a proxy fails to capture the full space of optimal design trade-offs, called the *Pareto set.* A design point is Pareto-optimal if an adjustment to the design cannot simultaneously improve *all* performance metrics; any improvement to one metric necessarily worsens another. If $F(x) = (f_1(x), f_2(x))$ is a function that maps the *design space* onto the *performance space*, then F maps the Pareto *set* onto the Pareto *front.* While the main task of a designer might be considered navigation of the Pareto front, its shape is typically nonlinear and even disconnected. Linear changes in α do not correspond to linear, or even continuous, changes on the Pareto front (see Figure 7-4). For this reason, guessing a reasonable α, or sampling over α, can be imprecise, unstable, or even intractable. Furthermore, the introduction of new performance metrics renders any previous choice of α obsolete.

Instead of using proxy objectives, we seek to discover, represent, parameterize, and explore the Pareto front directly. In our approach, the Pareto front discovery is done in a pre-computation step, and the resulting representation is used to define an interactive tool

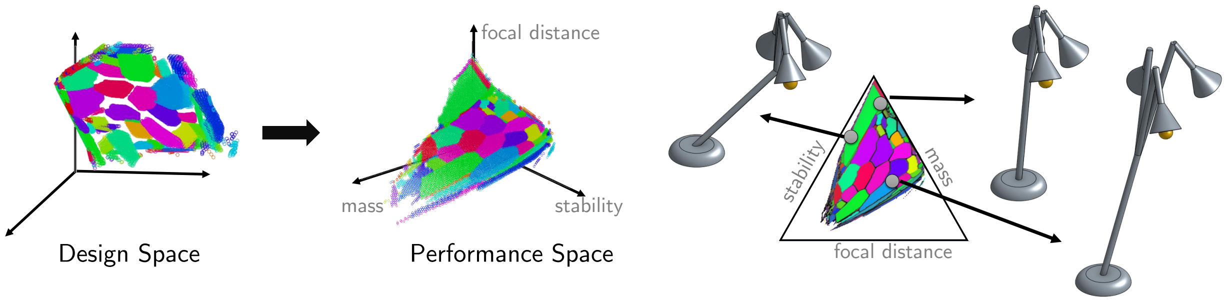


Figure 7-1: Our method allows users to optimize designs based on a set of performance metrics. Given a design space and a set of performance evaluation functions, our method automatically extracts the *Pareto set*—those design points with optimal trade-offs. We represent Pareto points in design and performance space with a set of corresponding manifolds (left). The Pareto-optimal solutions are then embedded to allow interactive exploration of performance trade-offs (right). The mapping from manifolds in performance space back to design space allows designers to explore performance trade-offs interactively while visualizing the corresponding geometry and gaining an understanding of a model's underlying properties.

that allows users to navigate the complex trade-offs between multiple performance metrics and instantaneously find corresponding designs. The main challenge in creating this representation is that finding all Pareto points requires on the one hand exploring the *diverse* solutions that correspond to different trade-offs and on the other hand *converging* to solutions in each particular direction to find points that are optimal in the Pareto sense.

From a technical perspective, our algorithm is built upon a first-order approximation of the Pareto front derived from duality theory in multi-objective optimization. This approximation serves two roles. First, it enables exploration of the Pareto front near a single point once it has been discovered. Second, it efficiently captures a *region* of the Pareto front, allowing us to approximate the entire front with just a few continuum pieces instead of a dense set of sample points. Each continuum piece also stores a mapping to the Pareto set in the design space. The end result is a technique that discovers the Pareto front and represents it as a union of relatively few manifold pieces that can be stored and queried efficiently within an interactive design tool that presents the designer with only the *relevant* (Pareto optimal) set of designs. Our representation is well-suited for the nonlinear and possibly disconnected nature of the Pareto front.

7.2 Related Work

The task of finding the set of optimal design trade-offs amounts to solving a multi-objective optimization problem, where the objectives are the performance metrics. Numerous methods have been proposed for solving multi-objective optimization problems. The Normal Boundary Intersection Das and Dennis [1998] and Normalized Normal Constraint Messac et al. [2003] methods aim to produce a well-distributed set of solutions, which can accurately approximate the shape of the Pareto front but can produce false positive solutions when the problem is non-smooth and/or contains local minima. Evolutionary algorithms often are applied to address this issue by iteratively modifying a population of candidate solutions that undergoes reproduction and removal similar to natural evolution; see Zhang and Xing [2017] for a survey.

The main difference between these methods and our approach is that instead of searching for a diverse set of discrete points, our proposed method provides a compact representation covering contiguous regions within the space of solutions. Our approach leverages the fact that while discovering individual points on the Pareto front can be difficult, locally searching around known solutions is easier. Using our method we obtain a relatively small set of manifolds whose union approximates the Pareto front. This representation provides an easily-navigable set of solutions in both design space and objective space, which can be applied to visualization and analysis.

7.3 Mathematical Preliminaries

In this section, we introduce the mathematical toolbox that formalizes the idea of trade-offs in an engineering context. This quick summary establishes the notation we need in our paper; for a broader discussion we refer the reader to Deb and Deb [2014].

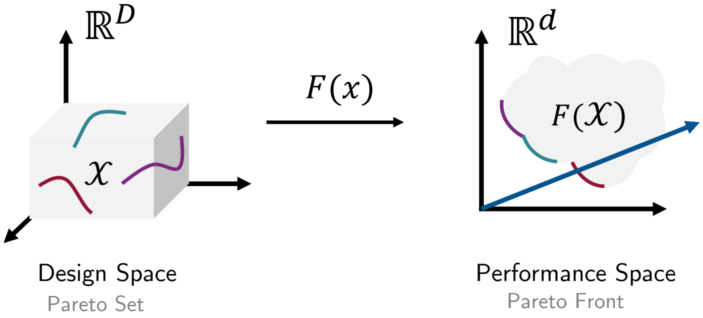


Figure 7-2: The Pareto set represents the points in design space with optimal performance trade-offs that get mapped to the Pareto front in performance space. Different colors indicate different manifolds in design and performance space with a one-to-one mapping. Any ray from the origin (blue line) can only intersect the Pareto front once.

7.3.1 Definitions

The set of exposed parameters for an engineering model is known as the *design space*:

Definition (Design space and constraint)**.** *The* design space $\mathcal{X}$ *for a multi-objective problem is defined as a subset of* $\mathbb{R}^D$ *of feasible points:*

$$\mathcal{X} := \{\mathbf{x} = (x^1, \ldots, x^D) \in \mathbb{R}^D : g_j(\mathbf{x}) \leq 0 \ \forall j \in \{1, \ldots, K\}\}.$$

Here, each function g_j represents a single constraint on $\mathbf{x}$*. We use $G(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^K$ to denote the concatenation $(g_1(\mathbf{x}), \ldots, g_k(\mathbf{x}))$.*

Intuitively, $\mathcal{X}$ is the set of all manufacturable objects. Constraints g_j might capture hard constraints identified by the engineer, such as a limit on the total material available to manufacture an object.

Next, we need a notion of an objective function for optimization:

Definition (Performance metric and space)**.** *A set of* performance metric *functions $f_i : \mathbb{R}^D \to \mathbb{R}$ assign real-values to each design vector* $\mathbf{x}$*; we use $F(\mathbf{x}) : \mathbb{R}^D \to \mathbb{R}^d$ to denote the concatenation $(f_1(\mathbf{x}), \ldots, f_d(\mathbf{x}))$. We choose the convention that* small *values of $f_i(\mathbf{x})$ are desirable for metric f_i. The* performance space *$\mathcal{S}$ is the image of the design space $\mathcal{X}$ under the performance metrics:*

$$\mathcal{S} := F(\mathcal{X}) \subseteq \mathbb{R}^d.$$

The performance metric functions are often complex and computationally intensive. Multi-objective problems typically involve multiple performance metrics ($d \geq 2$), e.g. weight, torque, and other measures; typically $d \ll D$.

Our algorithm reveals only those points in design and performance space that are not out-performed on every axis by some other design:

Definition (Pareto optimality)**.** *A point $\mathbf{x} \in \mathcal{X}$ is* Pareto optimal *if there does not exist any $\mathbf{x}' \in \mathcal{X}$ so that $f_i(\mathbf{x}) \geq f_i(\mathbf{x}')$ for all i and $f_i(\mathbf{x}) > f_i(\mathbf{x}')$ for at least one i. The set of all Pareto-optimal points is the* Pareto set *$\mathcal{P} \subseteq \mathbb{R}^D$; the image $F(\mathcal{P}) \subseteq \mathbb{R}^d$ is the* Pareto front*.*

7.3.2 KKT Conditions

Our algorithm pre-computes $F(\mathcal{P})$ and represents it compactly to allow for fast exploration and optimization. We represent $F(\mathcal{P})$ as a union of $(d-1)$-dimensional manifolds. As we will show, for (almost) any point $\mathbf{x} \in \mathcal{P}$, there is a local neighborhood $\mathcal{B}(\mathbf{x})$ such that $F(\mathcal{P}) \cap \mathcal{B}(\mathbf{x})$ is a $(d-1)$-dimensional manifold in $\mathbb{R}^d$; we store $F(\mathcal{P})$ as the union of a sparse set of simple manifold approximations.

Intuitively, this approximation is justified as follows. From the definition of Pareto optimality, if a point $x \in \mathcal{X}$ is Pareto optimal, then there is no other point whose performance is not worse than x in every metric and better than x in at least one metric. Hence, we can draw a ray in performance space along which there can be at most one Pareto optimal point, illustrated in Figure 7-2. This effectively extracts just points on the boundary of the Pareto front, a lower-dimensional set and justifies the following proposition:

Proposition 1. *For every nonnegative $\alpha \in \mathbb{R}^d$, there exists at most one $t > 0$ such that $t\alpha \in \mathcal{P}$.*

We parameterize this set using a "performance buffer," defined in §7.5.1.

This lower-dimensional observation is justified by the theory of *KKT conditions* from multi-objective optimization. Following Deb and Deb [2014], we denote Pareto-optimal points as solution of the *primal* problem notated

$$\begin{array}{rl} \min_x & \{f_i(x)\} \\ \text{s.t.} & x \in \mathcal{X}. \end{array} \tag{7.1}$$

Standard approaches to multi-objective optimization aim at finding solutions to this primal problem. The main challenge is that the space of solutions that are Pareto-optimal is large, disconnected, and prone to local minima. Typical approaches use genetic algorithms to find solutions that are diverse and optimal Zhou et al. [2011]. On a high level, these methods try to reach the Pareto front by searching in different directions and using randomization to avoid local minima.

Inspired by *primal-dual algorithms* in optimization, the key insight in our approach is that while discovering a single point on the Pareto set is challenging, once a point has been found it can be used to uncover a large Pareto region on its neighborhood. Our approach is to consider a *dual* problem, defined by the so-called KKT conditions:

Proposition 2 (KKT conditions Hillermeier [2001]). *Assuming that f_i and g_k are continuously differentiable and that the vectors $\{\nabla g_{k'}(\mathbf{x}^*) \mid k' \text{is an index of an active constraint}\}$ are linearly independent, then for any solution $\mathbf{x}^*$ to Equation 7.1 there exist dual variables $\alpha \in \mathbb{R}^d$ and $\beta \in \mathbb{R}^K$ such that*

$$\left\{\begin{array}{l} \mathbf{x}^* \in \mathcal{X} \\ \alpha_i \geq 0 \quad \forall i \in \{1, \ldots, d\} \\ \beta_k \geq 0 \quad \forall k \in \{1, \ldots, K\} \\ \beta_k g_k(\mathbf{x}^*) = 0 \quad \forall k \in \{1, \ldots, K\} \\ \sum_{i=1}^{d} \alpha_i = 1 \\ \sum_{i=1}^{d} \alpha_i \nabla f_i(\mathbf{x}^*) + \sum_{k=1}^{K} \beta_k \nabla g_k(\mathbf{x}^*) = 0 \end{array}\right\} \tag{7.2}$$

It is worth noting that KKT conditions are necessary but not sufficient, so we must *a posteriori* check that candidate points satisfying these conditions are indeed Pareto-optimal. This check is extremely efficient thanks to our performance buffer. The KKT conditions verify that at least locally the Pareto front is $(d-1)$-dimensional, thanks to the constraint on the sum of α.

7.4 First-Order Approximation

We begin our technical discussion by motivating a first-order approximation of the Pareto front. This formula is a straightforward corollary of the KKT conditions in Proposition 2; conceptually, it characterizes the proper directions to walk in design and performance space to maintain Pareto optimality after a single Pareto point is found. This formula can be understood as a source of efficiency for our algorithm relative to other sampling algorithms, since entire neighborhoods rather than individual points on the Pareto front are captured.

We state our condition as follows:

Proposition 3 (KKT Perturbation). *Suppose $\mathbf{x}(t) : (-\varepsilon, \varepsilon) \to \mathbb{R}^D$ is in the Pareto set in a neighborhood of $t = 0$, that is, $\mathbf{x}(t) \in \mathcal{P}$ for all $t \in (-\varepsilon, \varepsilon)$. Taking α and β to be the KKT dual variables corresponding to $\mathbf{x}^* := \mathbf{x}(0)$, under the assumptions from Proposition 2 we have*

$$H\mathbf{x}'(0) \in \operatorname{Im}(DF^\top(\mathbf{x}^*)) \oplus \operatorname{Im}(DG_{K'}^\top(\mathbf{x}^*)). \tag{7.3}$$

where

$$\mathrm{H} := \sum_{i=1}^{d} \alpha_i H_{f_i}(\mathbf{x}^*) + \sum_{k=1}^{K'} \beta_k H_{g_k}(\mathbf{x}^*).$$

Furthermore,

$$DG_{K'}(\mathbf{x}^*)\mathbf{x}'(0) = 0. \tag{7.4}$$

Here H_u and D_u represent the Hessian and Jacobian of a function u, respectively; $DG_{K'}$ indicates the part of the Jacobian DG corresponding to the $K' \leq K$ active constraints. We prove this proposition in Appendix B.

Generically, these define a $(d-1)$-directional space of local exploration directions around $\mathbf{x}^*$. In particular, assuming H is invertible, (7.3) shows that $\mathbf{x}'(0)$ is in a $(d+K'-1)$-dimensional space; the -1 comes from the last line of (7.2), which effectively shows that the row spaces of DF and $DG_{K'}$ are linearly dependent. Equation (7.4) reduces the dimensionality by K', as needed.

7.5 Pareto Front Discovery

We now define an iterative algorithm for exploring the Pareto front, constructed from the above perturbation formula. Typical iterative approaches seek a diverse, dense set of solutions (points) that together approximate the Pareto front. In contrast, our algorithm represents the front piecewise-continuously as a set of manifolds. Our algorithm is therefore less sensitive to the uniformity of discovered points, so long as the manifolds expanded from these points completely cover the Pareto front. Instead, the distribution of points in our discovery algorithm will depend on the varying sizes and locations of the generated manifolds in objective space.

7.5.1 Data Structure

Assuming that performance metrics are positive, any point in the Pareto front will intersect a positive ray traced from the origin. From Proposition 1, any such ray will intersect at most one point in the Pareto front. Further, a Pareto point that intersects a given ray will have minimal distance to the origin when compared to all other points in performance space $\mathcal{S}$ that intersect this ray.

We therefore define the performance buffer as a $(d-1)$-dimensional array discretized using (hyper)spherical coordinates (see Figure 7-3). Inspired by the z-buffer used in rendering, a basic implementation of the performance buffer stores at each cell the point with minimum distance to the origin that intersects its corresponding ray. The performance buffer is updated at each iteration of the discovery algorithm, as new regions of the performance space are found.

To reduce stochasticity of our final result, in practice we extend the basic implementation by storing a list of *candidate solutions* at each buffer cell $B(j)$, instead of storing only the single solution that has minimal distance to the origin. These solutions are included if their distance to the origin is within an allowed tolerance δ_B of the minimal distance over all solutions in $B(j)$. Maintaining this set of solutions is useful for extracting a *sparse*

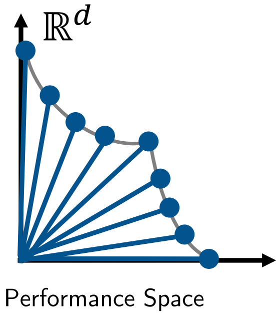


Figure 7-3: The performance buffer: since a ray from the origin can only intersect one point in the Pareto front, we use a buffer discretized by (hyper)spherical coordinates for storage.

approximation of the Pareto front (§7.5.4), which may forego choosing the closest sample to the origin at some buffer cells in favor of a simpler set of manifolds covering the front. For performance, we bound the number of stored solutions per cell, keeping only the top K ($K = 50$ for all experiments).

7.5.2 Discovery Algorithm

Overview We address diversity and convergence with an iterative procedure to discover the Pareto front. The algorithm is composed of three main steps (see Algorithm 2 and Figure 7-4). The first step is a *stochastic sampling* scheme that selects samples $\mathbf{x}_s^i$, $i = 1, \dots, N_S$ in the design space $\mathcal{X}$ (§7.5.2). The second step is a *local optimization* procedure that tries to push each sample $\mathbf{x}_s^i$ to a solution $\mathbf{x}_o^i$ on the Pareto set (§7.5.2). For each sample $\mathbf{x}_s^i$, a search direction $\mathbf{s}(\mathbf{x}_s^i) \in \mathbb{R}^d$ is selected to drive the local optimization scheme. Diverse directions are used to find a set of solutions that cover the different regions of the Pareto front. Finally, a *first-order approximation* of the Pareto front is extracted around $\mathbf{x}_o^i$ (§7.5.3). The perturbative formula in §7.4 is used to generate the $D \times (d-1)$ matrix M^i that defines an affine subspace $\mathcal{A}^i$ in design space that goes though $\mathbf{x}_o^i$.

The resulting manifold $F(\mathcal{A}^i)$ in performance space is then projected onto the buffer. If a point on $F(x^i) \in F(\mathcal{A}^i)$ is projected onto the buffer cell $B(j)$ and this point is considered a candidate according to the tolerance δ_B, then the buffer is updated. Each cell j on the buffer stores a list of solutions, each of which contains the point in design space, the corresponding map to performance space, and the corresponding affine subspace $\{(\mathbf{x}^i, F(\mathbf{x}^i), \mathcal{A}^i)\}_i$. The algorithm terminates if the buffer cells' average distance to the origin is not improved by δ_I after N_T iterations.

Stochastic Sampling We use stochastic sampling to initialize each iteration of the algorithm to avoid local minima. Since the performance buffer stores the current approximation of the Pareto front, we use these points as initial guesses. We uniformly sample buffer cells at random. For each selected cell j, we take the point $\mathbf{x}^j$ with minimal distance to the origin and perturb it as follows:

$$\mathbf{x}_s = \mathbf{x}^j + \frac{1}{2^{\delta_p}}\mathbf{d}_p$$

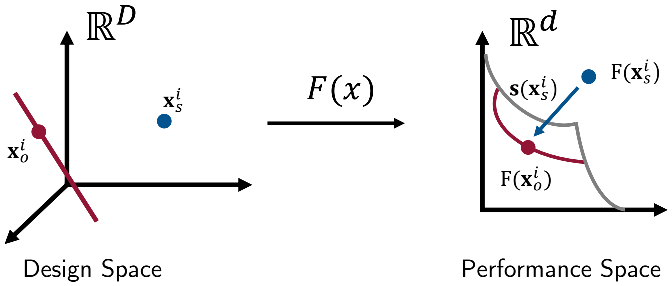


Figure 7-4: A single iteration of the discovery algorithm: random samples $\mathbf{x}_s^i$ are generated from the current data on the buffer (illustrated in gray) and optimized for a search direction $\mathbf{s}(\mathbf{x}_s^i)$ (blue arrow). A first-order approximation around the result of this optimization, $\mathbf{x}_o^i$, generates the corresponding manifolds in both design and performance space (red lines) and the buffer is updated based on this new data.

where $\mathbf{d}_p$ is a uniform random unit vector that defines a stochastic direction and δ_p is a uniform random number in $[0, \delta_P]$ used for scaling. The exponential factor trades off between exploration and exploitation—small scaling factors are typically preferred to explore local neighborhoods, but occasional larger values are desired to diversify the solutions. We clamp the result to ensure $\mathbf{x}_p \in \mathcal{X}$. In the first iteration, points are sampled uniformly from $\mathcal{X}$.

Local Optimization Each of the sampled points $\mathbf{x}_s$ is then optimized for Pareto optimality. A scalarization scheme is used to convert this multi-objective optimization problem into a single-objective problem which can be solved for each point. Previous work on multi-objective optimization proposes assorted scalarization functions that diversify the solutions across the Pareto front Das and Dennis [1998]; diversification is essential to avoid having solutions cluster in certain areas, failing to provide a good representation for the shape.

We find that the following scalarization function is most effective in our applications:

$$\mathbf{x}_o = \arg\min_{\mathbf{x}\in\mathcal{X}} \|F(\mathbf{x}) - \mathbf{z}(\mathbf{x}_s)\|^2 \tag{7.5}$$

where $\mathbf{z}(\mathbf{x}_s) \in \mathbb{R}^d$ is a reference point defined for each sample. This quadratic expression is inspired by previous work Zeleny [1973] and allows for discovery of solutions on non-convex regions of the Pareto front.

We use the performance buffer discretization to specify a unit-length search direction $\mathbf{s}(\mathbf{x}_p)$ for pushing $\mathbf{x}_s$ towards the Pareto front (see Figure 7-5). This suggests choosing the reference point $\mathbf{z}(\mathbf{x}_p)$ as:

$$\mathbf{z}(\mathbf{x}_s) = \mathbf{x}_s + \mathbf{s}(\mathbf{x}_s)C(\mathbf{x}_s), \tag{7.6}$$

where $C(\mathbf{x}_s) = \delta_s\|\mathbf{x}_s\|$ is a scaling factor depending on the distance to the origin. This scaling factor is important for diversity since setting the reference point too far from the Pareto front will make results cluster around specific solutions.

As shown in Figure 7-5, the buffer discretization defines a search direction for each buffer cell j. For further diversity, instead of setting $\mathbf{s}(\mathbf{x}_s)$ as the search direction for the buffer cell j

ALGORITHM 2: Pareto set discovery given performance metrics F and design constraints that define $\mathcal{X}$.

```
$B$: performance buffer array;
$B(i) \leftarrow \emptyset, \forall i$;
do
    $\mathbf{x}_s^0, \ldots, \mathbf{x}_s^{N_s} \leftarrow$ stochasticSampling($B$, $F$, $\mathcal{X}$);
    forall $\mathbf{x}_s^i$ do
        $D(\mathbf{x}_s^i) \leftarrow$ selectDirection($B$, $\mathbf{x}_s^i$);
        $\mathbf{x}_o^i \leftarrow$ localOptimization($D(\mathbf{x}_s^i)$, $F$, $\mathcal{X}$);
        $M^i \leftarrow$ firstOrderApproximation($\mathbf{x}_o^i$, $F$, $\mathcal{X}$);
        updateBuffer($B$, $F(M^i)$);
    end
    if buffer not updated on past $N_i$ iterations then
        break;
    end
while within computation budget;
Return $B$;
```

where $\mathbf{x}_s$ get projected, we select the search direction assigned to a cell on the neighborhood of cell j selected uniformly at random. The neighborhood of a cell is defined by all the cells that are within distance δ_N.

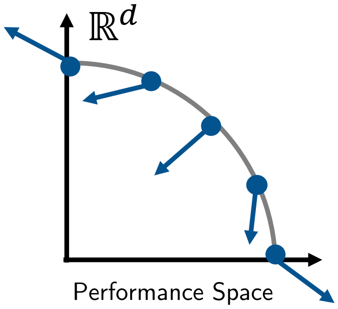


Figure 7-5: Search directions for buffer cells for $d = 2$. For diversity, different regions of the performance space get assigned different search directions. We use the buffer discretization to define these directions as illustrated in the figure.

7.5.3 First-Order Approximation

For each point $\mathbf{x}_o^i$, we use the result in Proposition 3 to find $d - 1$ directions for local exploration stored in a matrix M^i.

As discussed in §7.4, equations (7.3) and (7.4) generically define a $d - 1$ dimensional space. In practice, however, one must consider two special cases: (1) when $\mathrm{Im}(DF^\top(\mathbf{x}^*)) \oplus \mathrm{Im}(DG_{K'}^\top(\mathbf{x}^*))$ is low rank and (2) when H is low rank.

The first special case occurs when two performance objectives agree. In this case, the Pareto front locally will be represented by a manifold with dimensionality smaller than $d-1$. Consider, for example, a two-objective problem where the performance metrics are the weight and material cost of a single-material model. Since these objectives are not conflicting, the Pareto front is defined by a single point where the volume is minimized. Typically this is not the case for interesting problems in multiobjective optimization where the challenge is due to conflicting objectives. Therefore, in our implementation, we assume that this does not happen.

The second special case is more common, since it results from having design variables that do not affect the performance. While design variables that do not affect the performance at any configuration can be easily discarded in a pre-processing step, it is common to have design variables that have overall impact but locally are ineffective. In such cases, equations (7.3) and (7.4) define a space with dimensionality higher than $d-1$. This means that locally the Pareto *set* (design space) has higher dimensionality. Since the Pareto *front* (performance space) can never have dimensionality greater than $d-1$ (see Proposition 1), however, this means that there are multiple affine subspaces that locally map to the Pareto front. In our implementation, we deal with these cases by selecting $d-1$ directions uniformly at random.

Storage Given M^i, which defines an affine subspace around $\mathbf{x}_o^i$, we find an orthonormal frame and uniformly sample on a grid defined by this frame in design space. We set the grid size to be large enough to reach the boundaries of $\mathcal{X}$ and discard points that are not in $\mathcal{X}$. We then map all valid points to performance space using F and project the results onto the buffer. For $d = 2$, this projection is done by interpolating line segments, and for $d = 3$, we define a triangle mesh and use barycentric coordinates for interpolation.

As previously discussed, the buffer stores all of the solutions that are within a given tolerance. For each solution $(\mathbf{x}, F(\mathbf{x}), \mathcal{A})$ mapped to a cell j in the buffer we compare its distance to the origin with the minimal distance stored in the solutions for cell j. If the result is within δ_B, we append it to the solution list for that cell; otherwise it is rejected. If the solution is closer to the origin than any other solution on the buffer, we traverse the list rejecting all candidate solutions that are no longer within tolerance.

7.5.4 Sparse Approximation

After the Pareto front has been discovered, the final step is to select for each cell on the buffer a single solution from the list of candidate points. Our goal is to assign a unique value to each buffer cell to minimize discontinuities in design space while maintaining optimality within tolerance. In cases with many design variables, it is possible to have more than one solution in design space that maps to the same point in the Pareto front. We aim at selecting between these solutions so that adjacent buffer cells map to solutions that are close in design space. Further, we want a sparse set of first-order approximations that accurately represents the Pareto front.

Since points that are represented by the same first-order approximation are close in design space, we can optimize for both of these objectives by solving a labeling problem. Each label l^i corresponds to a linear subspace $\mathcal{A}^i$ on the set of spaces found by the discovery algorithm. Our goal is to choose a label for each buffer cell j so that: (1) the label of a cells is similar to the label of its neighbors and (2) the assigned label for a given cell is on the list of solutions $B(j)$, with priority given to solutions with smaller distance to the origin. This can be expressed and solved as a graph-cut problem.

Compared to an approach that takes the best value in each buffer cell, our graph-cut algorithm finds a sparse set of first-order approximations, providing local continuity at the expense of additional approximation error. We define the approximation error $e_A(j, i)$ associated to assigning label l^i to cell $B(j)$ as the difference between the distance to the origin of the candidate solution on $\mathcal{A}^i$ and the minimal distance to the origin of all solutions in $B(j)$. The graph-cut formulation aims at segmenting the buffer into large continuous regions while minimizing this error. From the buffer construction, the error is bounded by a user-defined tolerance, δ_B. In typical applications, engineering safety factors should be used to determine this tolerance.

We use the technique described in Boykov et al. [2001] and the provided implementation. The unary term $E_U(j, i)$ is set to $e_A(j, i)/\delta_B$ if $\mathcal{A}^i$ is on the list of candidate solutions for $B(j)$ and C_{inf} otherwise. The binary term $E_B(j, k)$ is set to 1 for every point if j and k are within a δ_R neighborhood from each other. In our experiments, we set $C_{\text{inf}} = 10$ and $\delta_R n = 2$. A post-processing step is performed to filter out outliers.

7.5.5 Visualization

The performance buffer provides a discretization of the points on the Pareto front but may also contain points that are not Pareto-optimal. The final step of the algorithm is to remove all buffer cells that fall into this latter category. This can be done by simply checking for dominance based on Definition 7.3.1.

For visualization, we embed the buffer in a $(d-1)$-dimensional space to allow for easy exploration (see Figure 4-1). For $d = 2$, the embedding is a line, and for $d = 3$ the embedding is a triangle. In both cases, the extreme points correspond to maximizing a given performance metric. Since we handle minimization problems, each metric is optimized at the opposite vertex for $d = 2$ or edge for $d = 3$. We define a color map for the embedding to highlight different affine subspaces. We color the embedded shape assigned to each point as a hue based on the corresponding affine subspace and a value based on the distance between neighbors. This allows us to see the transitions in the design space.

The pre-computed one-to-one mapping between piecewise linear regions in design space and their corresponding manifolds in performance space allows us to generate the geometry that corresponds to each performance trade-off in real time, allowing for interactive navigation of this embedded space.

7.6 Results

We have implemented the algorithm above and run experiments for $d = 2$ and $d = 3$ with design parameters varying from $D = 3$ to $D = 21$ depending on the problem. For all of the experiments, we rescale the problems so that both $\mathcal{X} \in [0, 1]^D$ and $F(\mathcal{X}) \in [0, 1]^d$ and use the same parameter settings for all experiments: $\delta_B = 10^{-2}$, $\delta_I = 10^{-4}$, $\delta_S = 0.3$, $\delta_P = 10$, and $\delta_N = 0.2|B|$, where $|B|$ is the buffer size.

7.6.1 Experiments

We test the proposed algorithm against a set of well-established benchmark problems for multi-objective optimization, each of which covers a number of criteria for discovering the Pareto front.

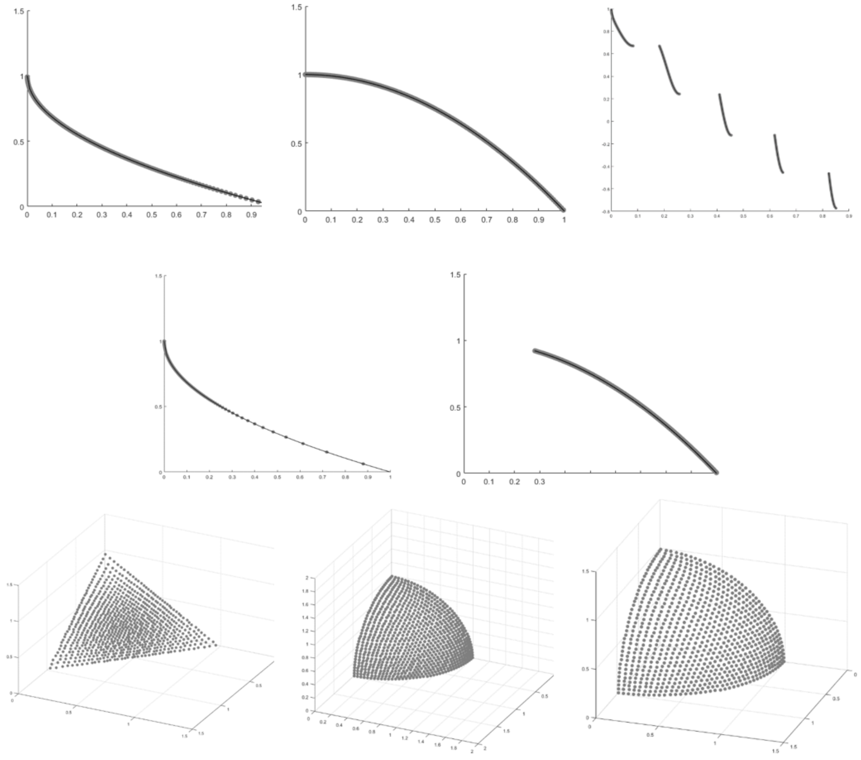


Figure 7-6: Nondominated solutions for various well-established benchmark problems using our proposed approach. Top two rows: Solutions for the five real-valued ZDT problems. Bottom row: Solutions for the first three DTLZ problems with three objectives. Our approach was able to converge to the ground truth Pareto front in all cases.

The ZDT test suite Zitzler et al. [2000] is perhaps the most widely-applied test suite for multi-objective optimization. This is due largely to the fact that the five real-valued tests in the set cover a broad range of geometric forms in both the Pareto front and Pareto set (concave, convex, disconnected). Additionally, these tests highlight the difficulties of multimodality (presence of multiple local minima) and, because they have well-defined optimal solutions, they are easily verifiable. The DTLZ test suite Deb et al. [2002] offers little new in the way of geometric complexity, but it does provide the option to optimize for more than two objectives. This is an especially rare attribute for well-established test suites and is a crucial requirement for the scope of our project. In particular, the reference point mapping and manifold generation become less intuitive for higher dimensional problems. For this reason, we use the DTLZ suite to validate and visualize the results of our method in three dimensions.

Figure 7-6 shows the results of the points stored in the performance buffer over the true front which is known for these test functions, validating that our proposed method manages to converge to the correct solution for these examples.

These test functions provide an empirical validation that our method can avoid local minima, discovering the true front even for multi-objective optimization problems that are designed to be challenging. Compared to state of the art methods for Pareto front discovery (according to a recent survey Zhang and Xing [2017]), our method generates a collection of manifolds, as opposed to point samples on the front (see Figure 7 of Zhang and Li [2007], which contains an identical experiment to our Figure 7-6). The computation time, measured by the number of function evaluations, is at least comparable to the state of the art. For example, Zhang and Li report ~ 10000 function evaluations for ZDT1 with 15 parameters, while our method uses only 6100 function evaluations. In addition, our final representation is more compact—since all ZDT benchmark examples can be represented by a single manifold—and is amenable to parallelization.

Since the Pareto set for each of the ZDT and DTLZ functions lies on a single affine subspace, our method recovers the entire front after finding a single solution that is Pareto-optimal. Therefore, to stress-test our approach and demonstrate that it can generically approximate the Pareto front by piecewise linear regions in design space, we test our method on a "Fourier benchmark": functions defined by linear combinations of sines. We define each performance metric f_j as

$$f_j(\mathbf{x}) = \sum_{i=1}^{D} \sum_{k=1}^{K} \alpha_{i,k} \sin(kx_i + B_{i,k}), \tag{7.7}$$

where $\alpha_{i,k}, \beta_{i,k} \in [0,1]$ are selected uniformly at random. We ran experiments for varying numbers of design and performance parameters. The results in Figure 7-7 illustrate the different affine subspaces that are used to construct the Pareto fronts, denoted in different colors.

Figure 7-8 compares our solution to a method that first discovers Pareto-optimal points and then uses a piecewise-linear interpolation in design space. Our approach, in addition to discovering points on the front more efficiently, has the critical advantage of being faithful to the topology of the front and its relationship to the preimage in design space (the Pareto set). As shown in the figure, there is no guarantee that interpolating the preimages of two solutions adjacent in objective space (blue points) will produce another Pareto-optimal solution (red points). Our method for generating space approximations (Section 7.5.4) automatically detects these discontinuities in the Pareto set.

The buffer construction and graph-cut algorithm guarantee that the results of this sparse approximation are optimal within a tolerance (δ_B) if each candidate point on $B(j)$ that has minimal distance to the origin is on the true Pareto front. This result, however, depends not only on our ability to avoid local minima, which was empirically validated on the ZDT and DTLZ benchmarks, but also on the first-order approximation. The quality of the first-order approximation depends on how well the Pareto set can be approximated by a linear function. Since the Pareto sets of the ZDT and DTLZ benchmarks are linear, we quantitatively evaluate the proposed local linear approximation on the Fourier benchmark. The result is shown in Figure 7-9, which, illustrates how far away the optimal candidate point $\mathbf{x}_j^*$ on each buffer cell is from the actual Pareto front.

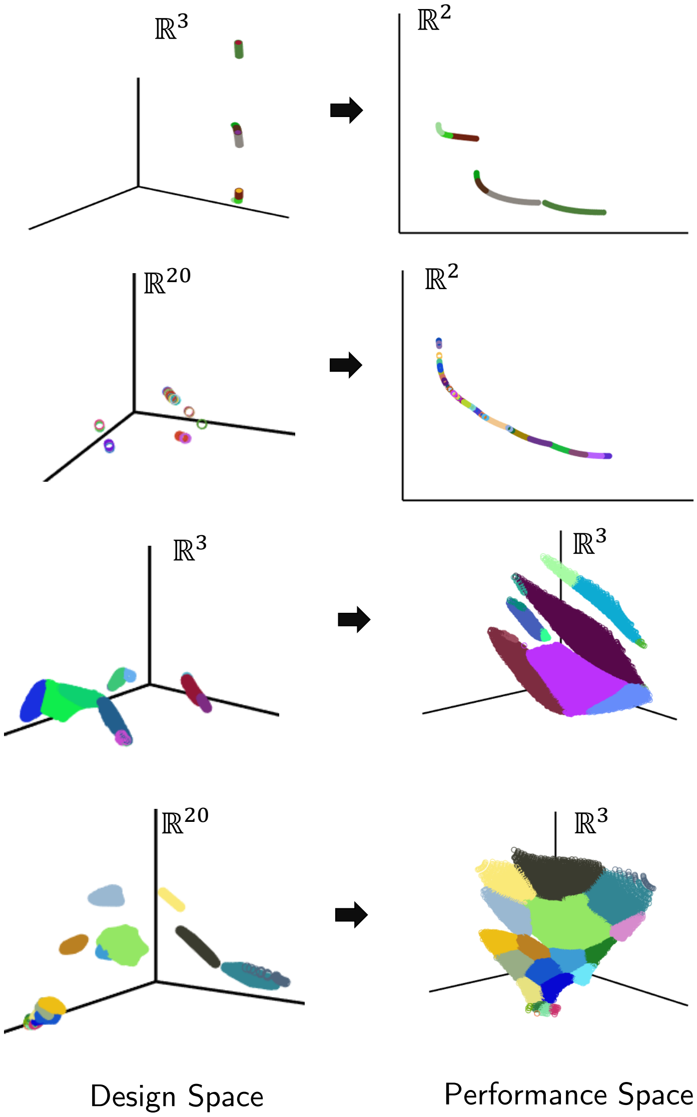


Figure 7-7: Results for Pareto front discovery on Fourier benchmark. The figure illustrates how the Pareto front can be covered by a set of individually smooth regions which are mapped via F from affine subspaces in the design space. Each corresponding pair in design and performance space is illustrated in a different color. In high-dimensional cases in design space, dimensionality-reduction via principal component analysis is applied for visualization purposes.

To measure this distance, we ran an additional local optimization that tries to push each point $\mathbf{x}_j^*$ the direction $\mathbf{d}_N^j$ normal to the front at $F(\mathbf{x}_j^*)$:

$$\min_{\mathbf{x}\in\mathcal{X}} \left\| F(\mathbf{x}) - \left(F(\mathbf{x}_j^*) + \delta_N \mathbf{d}_N^j\right) \right\|, \tag{7.8}$$

where we set $\delta_N = \delta_B = 10^{-2}$. Figure 7-9 shows that the approximation error (distance in performance space) for the problem showed in the the first row of Figure 7-7 is below

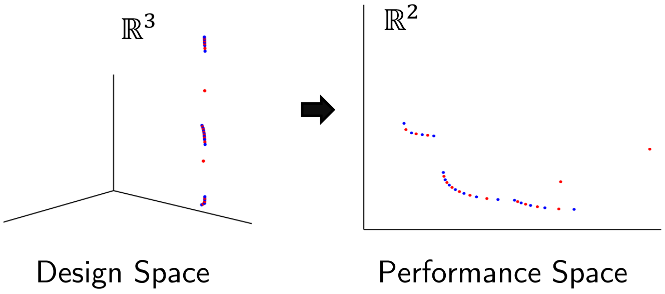


Figure 7-8: A direct piecewise-linear interpolation result of the example shown on the first row of Figure 7-7. The optimal solution is chosen from the list of points at each buffer cell (blue points) and a denser sampling is generated by linearly interpolating the preimage of neighboring points in performance space (red points). Since the Pareto front is comprised of distinct manifolds, the linearly interpolated points have no guarantee of Pareto optimality. For illustrative purposes, the interpolation is performed on a sparse set of the discovered solutions.

4.0×10^{-4} in the worst case and below 10^{-5} for over 97% of points. This result shows that for general functions defined by Fourier series, our method can robustly approximate the front. This is done by iteratively adding more local manifolds until the improvement on an average cell is bellow $\delta_I = 10^{-4}$ (stopping parameter). For the example in Figure 7-9 each buffer cell has, on average, 10.42 candidate points, which correspond to first order approximations that are within error $\delta_B = 10^{-2}$ of each other.

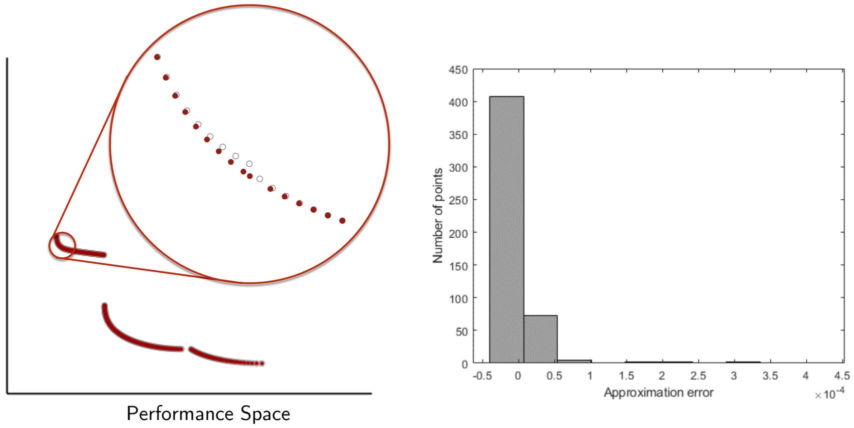


Figure 7-9: Performance Space Assessment of the approximation error associated with our first-order expansion method on the example shown on the first row of Figure 7-7. Left: plot in performance space of the discovered Pareto front (grey) and results of the same points after performing an additional local optimization (maroon). Right: histogram showing the approximation error for points on the discovered Pareto front.

Finally, we show the behavior of our first-order approximation on the bi-objective Kursawe problem Kursawe [1991], which has a disconnected, asymmetric Pareto front. Figure

7-10 shows the expanded curve using our algorithm (red) alongside a number of curves expanded in random directions (gray). Our generated direction in design space (black) approximates the shape of the true Pareto front in objective space better than its randomized counterparts that are not obtained using the perturbative formula.

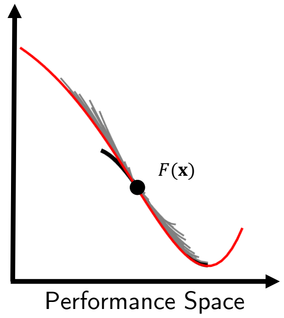


Figure 7-10: First-order approximation for a point $\mathbf{x}$ on the Pareto front of the bi-objective Kursawe problem Kursawe [1991] (true front shown in black). The first-order approximation defined by our method (red) is compared to the mapping of affine spaces around $\mathbf{x}$ generated using directions chosen uniformly at random (gray).

7.6.2 Design Applications

We additionally run experiments on several CAD models, each with a specified set of design and performance metrics. We use three models from Schulz et al. [2017c] that are assessed using a combination of pre-computed physical simulations and geometric analysis. We also experiment with higher-dimensional design spaces using a twelve-parameter boat parametrized by cage-based deformation Ju et al. [2005] and a twenty-one-parameter lamp designed in CAD. Figures 4-1 and 7-11 show the resulting solutions for each of these experiments in both design space and performance space, along with the corresponding embedding and (selected) mesh samples. We also implement a simple interface that provides an interactive visualization of geometries corresponding to each point in the embedding (see the supplemental video).

The top row of Figure 7-11 shows a wrench example with three design parameters: handle length, head thickness, and fillet radius (the rounding along the edges that connects the head to the handle). The measured performance metrics are stress for a given torque (maximum von Mises stress computed with FEA, implementation from Schulz et al. [2017c]), mass, and force required to generate a given torque. In this example, most of the front can be expressed as a single affine space where the fillet radius and the length are maximized. This result is due to the fact that these parameters have a large impact on minimizing the stress and force for a given torque, while the negative impact on the mass is largely negligible. Only in a very small region of the Pareto front do solutions corresponding to a small fillet radius appear. For these regions, the head and fillet radii are minimized and the optimal trade-offs between mass and force/torque are achieved by varying the length of the handle. While the design space for this model is low-dimensional (only three variables), it highlights the strength of our method in exposing relationships between design parameters and performance metrics

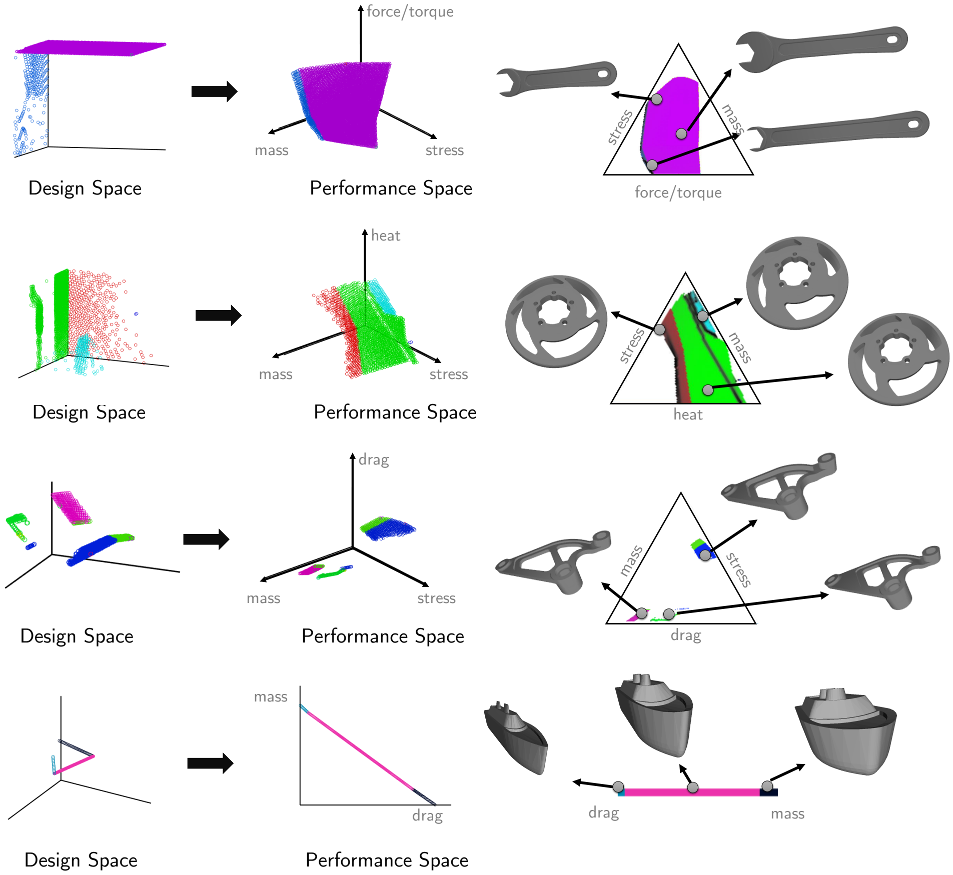
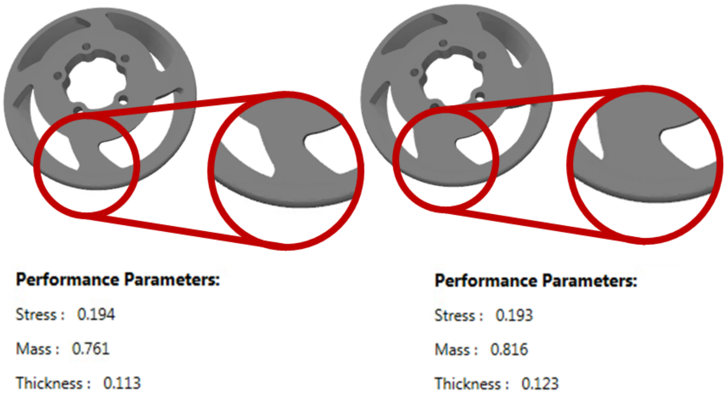


Figure 7-11: Examples of CAD models processed using our proposed technique. From left to right: Pareto-optimal points in design space (illustrated with multi-dimensional scaling via principal component analysis for models with more than three design parameters); Pareto-optimal points in performance space; the resulting embedding and illustrations of geometric results for some sampled points. Across all figures, the different colors correspond to different regions resulting from local expansion in design space.

that are not evident without analysis. Furthermore, it allows us to simplify the solution space by understanding that there are only two affine regions in design space which yield Pareto-optimal performance trade-offs.

The second row of Figure 7-11 shows a brake hub with three design variables: the angle of the inner hole, spoke thickness, and the thickness of the rim. The performance metrics for this model are stress (calculated by simulating the impact of directional forces and heat distribution), mass, and heat dissipation (approximated by the thickness of the rim). Our method highlights a strong discontinuity in design space represented by the black line that divides the green patch. Figure 7-12 shows two models on opposite sides of this divide. We observe that, while the performance parameters vary smoothly across this gap, the spoke thickness almost doubles. This result exposes a property of the measured stress which depends on both the heat distribution and directional forces from the brake; the reason why two very different designs can be so close in performance space is that the spoke thickness affects the heat distribution and the force-imposed object deformation in opposite ways. Our method exposes these different design configurations which, in fact, yield comparable performance results, providing engineers with a better understanding of the model's properties.



Performance Parameters:

Stress : 0.194

Mass : 0.761

Thickness : 0.113

Performance Parameters:

Stress : 0.193

Mass : 0.816

Thickness : 0.123

Figure 7-12: Example of two variations of the brake hub that have similar performance metrics but very different design parameters. Our system can expose these relationships providing intuition to designers about the structural nature of the model. The metrics are shown under a normalization for easy comparison—the Pareto front is rescaled to lie between zero and one.

The third row of Figure 7-11 illustrates a bike frame with four design variables that has been engineered to minimize mass, drag, and stress. What is interesting about this result is that the embedding is composed of a set of disconnected regions. This happens because of the geometry of the envelope of $F(\mathcal{X})$. Assuming that F is continuous and $\mathcal{X}$ is connected, the projection of $F(\mathcal{X})$ onto the buffer is also always connected. As previously discussed, however, not all points represented by the buffer are Pareto-optimal. For this example in particular, after we remove the suboptimal point from the buffer, we are left with a large empty region in the center of the embedding. These discontinuities in performance space reveal regions of trade-off where a small variation in design space will only slightly worsen one metric but significantly improve another. Such exposed properties can be very useful in aiding designers to decide between certain success metrics and define trade-offs.

The fourth row of Figure 7-11 illustrates a toy boat parameterized by a cage with twelve design variables and two performance metrics. The performance metrics are buoyancy (approximated by volumetric maximization) and drag from a frontal wind. This example shows how our method can find a locally smooth approximation of the Pareto front for high-dimensional design space. The resulting boats all have maximal length but the shape of the projection onto a frontal plane varies between solutions that represent different trade-offs between mass and drag.

Finally, Figure 4-1 illustrates a lamp with 21 parameters that was designed using a CAD package. Three parameters are used to define the position and orientation of each of the lamp's seven beams. The performance metrics for this model are: stability (measured by the distance of the projection of the center of mass to the center of the base), mass, and average distance between the lamp beams and a predetermined focal point that should be illuminated. As shown in the figure, our method returns an approximation of the Pareto front with many disconnected regions in the design space. This behavior arises due to the presence of many points in the design space that are mapped to the same regions on the Pareto front (i.e., there are many points in the Pareto front which can be locally approximated by manifolds of higher dimension than $d - 1$). An example of this is shown in Figure 7-13. This example highlights the utility of creating a sparse representation of the Pareto front with local manifolds. Since multiple points map to the same point on the Pareto front, a discrete approach would result in an approximation containing many disparate points across the design space. This discontinuous representation, however, provides little in the way of performance trade-off insights. Our method, on the other hand, creates locally smooth regions around areas of similar geometry. This configuration in turn allows designers to directly observe which parameters have a significant effect on local performance and which ones do not, a particularly useful feature in higher-dimensional cases.

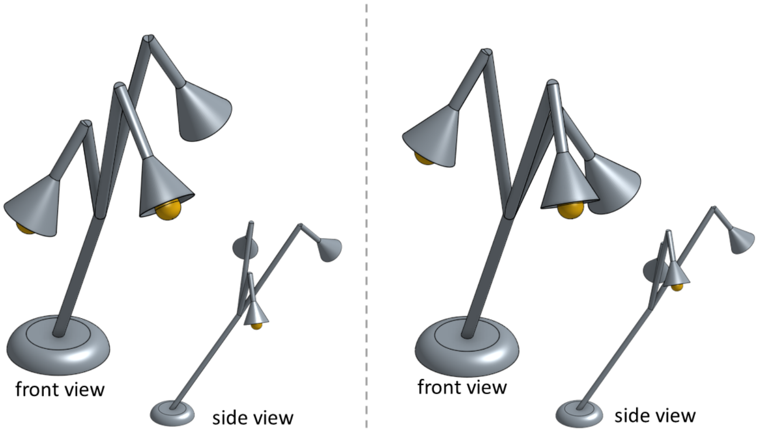


Figure 7-13: Example of two different lamps (front and side view) that have the same performance across all metrics. Due to the large dimensionality of the design space, these two configurations have the same stability, mass, and distance to the focal point.

7.7 Discussion

Real-world design problems can rarely be squeezed into one dimension. Instead, the process of engineering a physical object requires navigating a complex and potentially even disconnected space of candidate configurations. The algorithm and accompanying interactive tool presented in this paper represent a significant effort toward the larger goal of efficiently estimating and navigating the space of relevant designs for a given problem. Our technique efficiently reveals this space in a wide variety of scenarios, from benchmarks in multi-objective optimization to parameterized CAD models paired with expensive physical simulation tools. Its versatility indicates broad applicability across use cases in computational fabrication and beyond.

While our algorithm as-is can be plugged into many existing engineering pipelines, we also anticipate several avenues of research that can extend the basic model proposed here.

One important consideration comes from the human-computer interaction. Now that we can efficiently uncover and parameterize the Pareto front, what is the best way to display it to an engineer who must digest the space of candidate designs? Such a study can use our tool as a starting point, adjusting e.g. the embedding of the performance buffer to best reflect intuitive notions of proximity in performance space. On the opposite side of the spectrum between human interaction and automation, we also anticipate that our differentiable representation of the Pareto front can be incorporated into "hyperparameter" selection techniques that use a secondary function to choose between different points on the Pareto front.

Our study also suggests several intriguing mathematical and algorithmic challenges. While our benchmark study indicates that our algorithm yields a smooth and complete picture of the Pareto front—at least in realistic scenarios where the front is representable computationally—additional theoretical analysis could reveal convergence rates and/or the likelihood that our strategy will reveal the entire Pareto front. Of course, such theoretical analysis will probably require stronger assumptions on the performance metrics than are needed in practice, e.g. convexity or Lipschitz bounds, to rule out pathological cases; the challenge will be to select a theoretical model that is reflective of the scenarios we observe in our examples from CAD and engineering design. Further analysis could also extend the first-order approximation to handle cases in which the derivatives of the active constraints at a given point are themselves linearly dependent. Algorithmically, a clear next-step will be to extend our methodology to the regime where parameters are discrete or performance measures are not twice differentiable.

An additional challenge for the future spanning both technical and human-oriented aspects is to cope with higher dimensionalities for performance space. Currently our technique is designed for the $d = 2$ and $d = 3$ cases, for which the Pareto front is readily embedded in a display tool. Considering larger values of d will require several technical developments. For the sampling algorithm, the "curse of dimensionality" implies that sampling may take more iterations to converge; we anticipate that our manifold-based strategy will serve as a key component reducing computational burden in this regime by exploiting local structure to discover larger pieces of the front at a time. After uncovering the higher-dimensional Pareto front, we will also need to design a means of displaying the result in a fashion similar to Figure 7-11. While MDS embedding of the Pareto front may suffice, a more careful parameterization that preserves relevant relationships between the performance metrics may be desirable.

Even in the absence of these improvements, we anticipate incorporation of our exploration algorithm and visualization tool into software for CAD and 3D modeling. By helping engineers and designers understand the possible ways to trade off between performance metrics, we hope to alleviate the dependence on unintuitive and often brittle parameters currently permeating design software.

Chapter 8

Applications

8.1 Introduction

In this chapter we will discuss how the data-driven and performance-driven techniques we introduced in this thesis can be applied to develop new design tools. Such tools can be used by experts to design and manufacture products with increased performance in less time. As the advances in manufacturing increase the space of what can be made, such tools are essential to allow designers to efficiently navigate this complex space to create objects with integrated functionality. In addition, by increasing the capabilities of the design tools, these applications can bring down the barriers to entry for casual designers. By making it easier to compose designs and optimize them to meet specific needs, we can build systems that will hopefully pave the way to a new age of personal manufacturing and DIY design.

In what follows we will describe three efforts in building design tools for fabrications. We will first describe an end-to-end system for design and fabrication of robots with ground locomotion. We will then extend these ideas to optimization of multicopters. Finally, we will discuss a method for customization of carpentered items which can be fabricated with mobile robots.

It is important to highlight that these end-to-end systems not only validate the general approaches of this thesis, but they were also instrumental in inspiring the ideas presented in the previous chapters. While this thesis is organized by discussing first the general algorithms and techniques and then applications, the research was done concurrently—as evidenced by the publication date of the related publications. We have chosen application domains that can have direct impact and expose challenges and open problems. For example, in designing complex functional mechanisms such as robots and drones, we need to concurrently optimize both the geometry and also motion/control. In developing a carpentry design systems that have to interface with mobile robots fabrication process, we have to define new ways to constrain the design space appropriately and bring down the barriers to entry for casual designers.

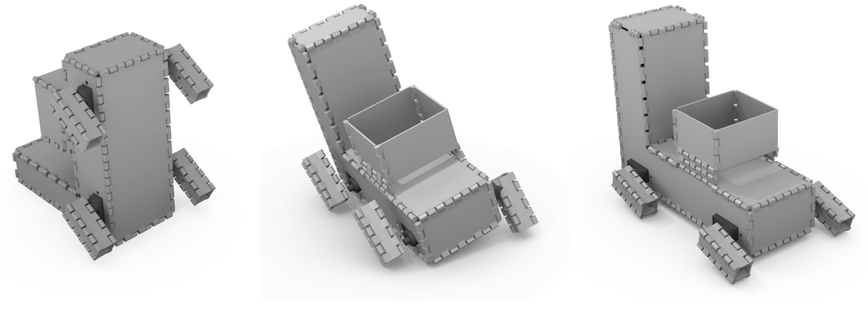
8.2 Interactive Design of Ground Robots

This system aims to democratize the design and fabrication of robots, enabling people of all skill levels to make robots without needing expert domain knowledge. Existing work in computational design and rapid fabrication has explored this question of customization for physical objects but so far has not been able to conquer the complexity of robot designs. We have developed Interactive Robogami, a tool for composition-based design of ground robots that can be fabricated as flat sheets and then folded into 3D structures. This rapid prototyping process enables users to create lightweight, affordable, and materially versatile robots with short turnaround time. Using Interactive Robogami, designers can compose new robot designs from a database of print and fold parts. The designs are tested for the users' functional specifications via simulation and fabricated upon user satisfaction.

8.2.1 System Overview

One of the main challenges in robot design is the interdependence of the geometry and motion. A typical design process for a robot starts with geometry design, followed by actuator selection, then controller design [Megaro et al., 2015b, Song et al., 2015]. However, minor changes in geometry can drastically simplify actuator or control design. Take for example the robot in Figure 8-1. This four-legged robot was designed by a user to deliver ice cream. In this task, speed is essential to ensure the ice cream does not melt. In addition, the robot must be steady enough that the ice cream does not spill on the way to its destination.

In general, the fastest way for a statically stable four-legged robot to move is for all the legs to move simultaneously, resulting in a scooting motion. With the user's initial design (Figure 8-1(a)), though, this gait is unstable and the robot topples backwards. The user can find a gait with the left and right pairs of legs moving sequentially that allows the robot to move forward with only a slight wobble (Figure 8-1(b)). However, changing the geometry slightly allows the robot to succeed with the original gait and maintain its high speed without falling over (Figure 8-1(c)), resulting in a robot that is almost 50% faster.



(a) Original
speed: 32.59 mm/s

(b) Modified gait
speed: 69.29 mm/s

(c) Modified geometry
speed: 99.76 mm/s

Figure 8-1: A robot design that topples while walking (a) can be modified to follow a gait that only wobbles slightly (b), but changing the geometry (c) allows the robot to move much faster and more steadily.

We present an interactive system that allows designers to explore how both geometry and motion affect robot performance, enabling them to make these types of design choices. Our system contains tools for geometry and gait design, as well simulations for evaluating the model (ref. Figure 8-2). Users interact with the tool through a graphical user interface (Figure 8-3) in which they can visualize the models they create and receive real-time feedback as to how changes to the design affect the robot's performance. The system keeps track of the geometry and motion in order to output a full fabrication plan once the user is ready to build the design.

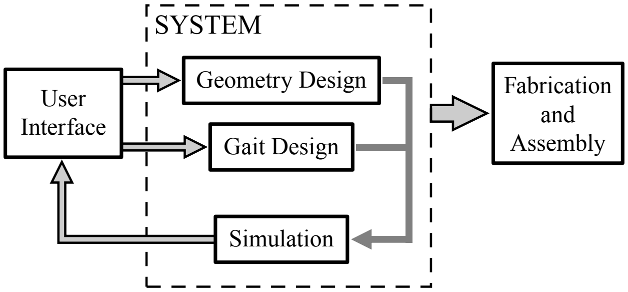
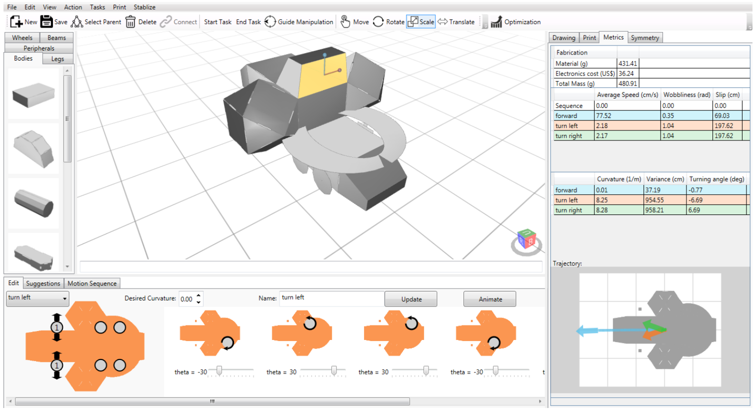

SYSTEM
User Interface
Geometry Design
Gait Design
Simulation
Fabrication and Assembly

Figure 8-2: System diagram. Users interact with geometry and gait design tools. The designed models are simulated to provide feedback to the user. The user may iterate over the design before fabricating and assembling the model.


File
Edit
View
Action
Tasks
Print
Stablize
New
Save
Select Parent
Delete
Connect
Start Task
End Task
Guide Manipulation
Move
Rotate
Scale
Translate
Optimization
Wheels
Beams
Peripherals
Bodies
Legs
Drawing
Print
Metrics
Symmetry
Edit
Suggestions
Motion Sequence
Update
Animate
Trajectory:

Fabrication	
Material (g)	431.41
Electronics cost (US$)	36.24
Total Mass (g)	480.91

	Average Speed (cm/s)	Wobbliness (rad)	Slip (cm)
Sequence	0.00	0.00	0.00
forward	77.52	0.35	69.03
turn left	2.18	1.04	197.62
turn right	2.17	1.04	197.62

	Curvature (1/m)	Variance (cm)	Turning angle (deg)
forward	0.01	37.19	-0.77
turn left	8.25	954.55	-6.69
turn right	8.28	958.21	6.69

turn left

Desired Curvature: 0.00

Name: turn left

theta = -30

theta = 30

theta = 30

theta = -30

Figure 8-3: User interface. Icons that link to geometry components are displayed on the left and the gait design tool is on the bottom. Users design models by dragging components into the center canvas and editing them. Performance metrics for the design are shown on the right.

Our design choices were driven by three main objectives: 1) facilitate user creativity while maintaining wide range of accessibility, 2) enable concurrent design of motion and geometry, 3) guarantee fabricability, and 4) provide real time feedback for ground locomotion design.

Since the design space of robots is too large to be explored interactively by casual users, we use data-driven design techniques to make the design space manageable. As discussed in Chapter 5, we introduce a component library and an assembly-based modeling tool to allow users to combine components in the database to create new robots. This library contains both geometry and gait component to allow concurrent design of motion and geometry. Because all components in the database have associated fabrication rules, the system can automatically output a full fabrication plan for the resulting robots.

In order to optimize for geometry and motion to meet desired design goals, users must be able to quickly compute relevant *performance* metrics such as speed or cost. Following the ideas discussed in Chapter 6, our system incorporates simulations and interactive feedback to aid users in making these evaluations.

8.2.2 Methods Overview

In what follows, we outline the main methods in our tool. For further details on the algorithms we refer the reader to [Schulz et al., 2017b].

Assembly-based Modeling Following the ideas discussed in Chapter 5, our assembly-based modeling tool allows users to create new designs by composing both geometric parts and motions from a collection of parametric components and manipulating their shape parameters.

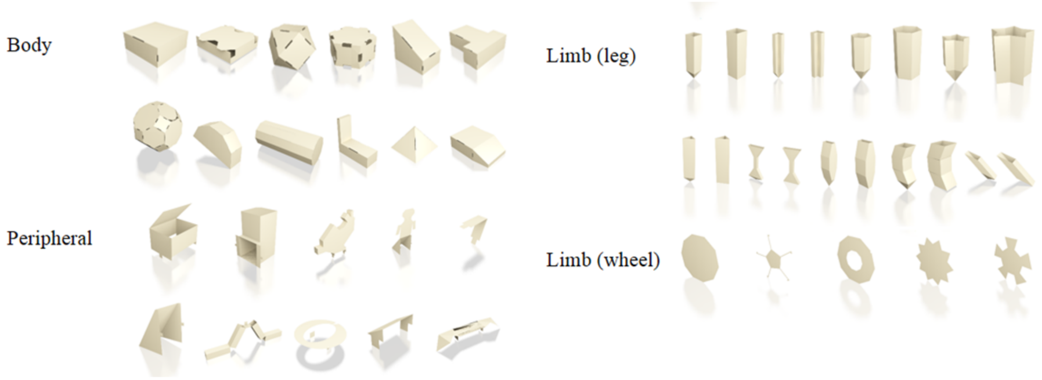
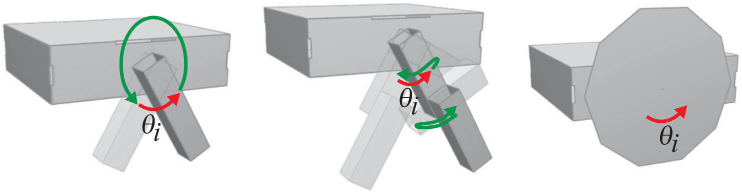


Figure 8-4: Geometry components by category. Our system's database contains 45 components: 12 bodies, 23 limbs, and 10 peripherals.

For this application, we use a linear parameterization that simplifies the data representation and speeds up computation without compromising expressiveness. Modeling is supported by a grammar that dictates composition rules To this end we drew many insights from the experience of robotics engineers. We categorized parts into bodies, legs, and peripherals based on the functionality of components in existing robot designs (see Figure 8-4). The joint controllers were chosen based on the physiology literature and parameterized according to when legs were expected to make contact with the ground [Ayyappa, 1997] (see Figure 8-5).

All the geometric components can be fabricated using our 3D print and fold method and the parametrization is designed so that this is preserved at any configuration (see Figure 8-6). The composition tool uses the parametric representation to snap geometric components together, constraining the composed models to a fabricable set of geometries. The composition rules are used to determine the electronic components needed in the assembly and the



(a) Single leg (b) Shoulder/Elbow (c) Wheel

Figure 8-5: Joint controllers for each joint type. Controllers for every joint are separated into a step phase (shown in green) and a reset phase (shown in red). Users modify gaits by changing the θ_i values for each joint and defining the step sequence.

parametric motion representation is used for automatic generation of the control software. This allows the system to automatically generate a full fabrication plan for every composed model (see Figure 8-7).

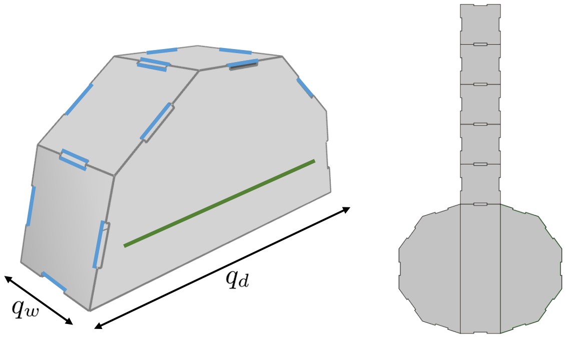
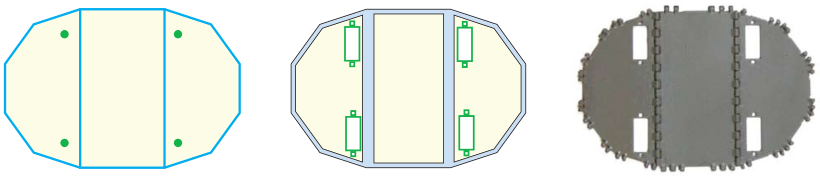


Figure 8-6: One of the geometry components in the database, with both its 3D shape and 2D unfolding. The component has parameter values of diameter q_d and width q_w. The 3D and 2D representations are coupled so that they are simultaneously updated as parameters are changed. In order to allow the assembly-based modeling to work with our fabrication method, each component is also annotated with a set of connecting patches that indicate where components are allowed to be connected together. The green line indicates a patch for functional connections, and the blue lines are patches for static connections.

Interactive Feedback Optimization and simulation tools are used to guide users in realizing a stable ground robot. Motion is simulated using forward kinematics under the user specified giants. The simulations are used to compute metrics that provide the user with information about the robot's expected performance. We chose metrics that are commonly of interest to roboticists designing ground behavior. These objectives allow inexperienced users to make more intelligent design tradeoffs than when using single-metric systems often found in the literature. In addition to fabrication cost, the system evaluates the following metrics, which tell users about the robot's overall performance: average speed, wobbliness (amount of orientation variation), average slip, angle of rotation (change in heading in one gait cycle), average curvature of each gait, and variance. All metrics on designed gaits are



(a) Input design (b) Process connections (c) Final print

Figure 8-7: Our system automatically generates a 3D mesh for a foldable design. (a) The original design, with blue edges indicating static connections and green dots indicating functional connections. (b) Connections are processed by shrinking faces to make room for hinges and adding holes for servo mounts. (c) Hinges are added and the result is a complete mesh that is 3D printed.

computed for the robot's steady-state behavior.

The performance and fabrication metrics are computed in real time when a user creates or modifies a robot and are displayed to the user (see Figure 8-3). Our system also provides guidance to the user on how to manipulate model dimensions. To activate this feature, users select a metric value to improve (i.e., increasing speed, decreasing wobble, decreasing slip, etc.) and turn guidance on. When they next choose a face on the model to scale or translate, the system displays arrows on the control axes indicating which direction to change the dimensions to improve that value (Figure 8-8).

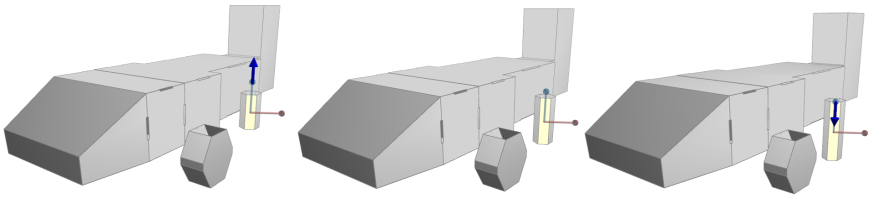
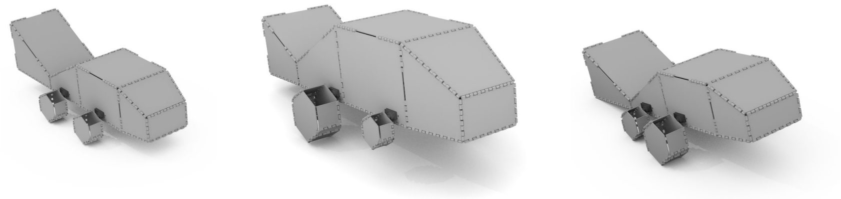


Figure 8-8: The system provides guidance arrows for users who want to make local geometry optimizations. The up and down arrows indicate that the user should lengthen and shorten the leg, respectively. No arrow indicates that the part dimension is already at a local optimum.

Finally, our system contains an automatic optimization method that will search for metric improvements in the full geometry space. The parametric representation defines the design space for a fixed topology over which optimization can occur. Users can only optimize one metric at a time (see Figure 8-9).

8.2.3 Results

To test the usability of our system and the effectiveness of the combined geometry and gait design approach, we introduced our system to eight users with no previous experience in robot design. The users were all engineering graduate students (three female, five male) between the ages of 22 and 31. Four of the students had previous experience with CAD and



(a) Original
$V = 25.59$ mm/s
$W = 0.46$ rad

(b) Maximize speed
$V = 52.28$ mm/s
$W = 0.75$ rad

(c) Minimize wobble
$V = 15.25$ mm/s
$W = 0.06$ rad

Figure 8-9: Optimization results for a four-legged fish robot for different metrics. Maximizing speed increases the overall size of the robot (b). Minimizing wobble makes the robot shorter (c).

modeling tools (SolidWorks, Blender, Maya, etc.). The users were given 20 min. of training in the system's features and were then asked to perform two tasks designed to evaluate the expressiveness of the geometry and gait design frameworks. In addition to these tests, we also fabricated six robot models designed in the system.

Geometry Design Figure 8-10 shows the models that the users created with the geometry composition tool. All of the models were functional vehicles that rolled forward without toppling. The results demonstrate a wide range of geometries that are achievable using our system. An enthusiastic user designed an additional 18 robots with different levels of complexity (Figure 8-11).

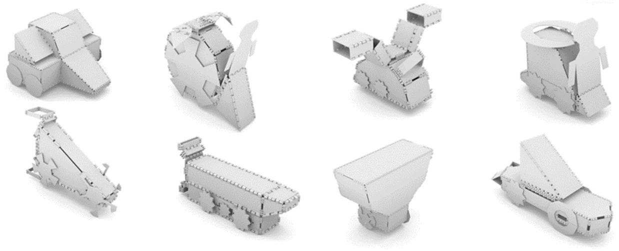


Figure 8-10: Cars designed by eight different novice users after a 20 min. training session with the tool. Users were given 10 min. to design their car.

Gait Design In the second task, users were given a robot design and asked to design a trajectory to navigate the robot through an obstacle course in the least amount of travel time. The users were given 15 min. to complete the task. Half of the users were allowed both to design gaits and to change the geometry parameters of the design, while the other half were restricted to gait design. Figure 8-12 shows the trajectories designed by each user and the total time of each gait sequence.

The robot geometry provided was similar to the one described in Figure 8-1, which topples during forward gaits where all legs have equal values for θ_i's. Users who were not allowed

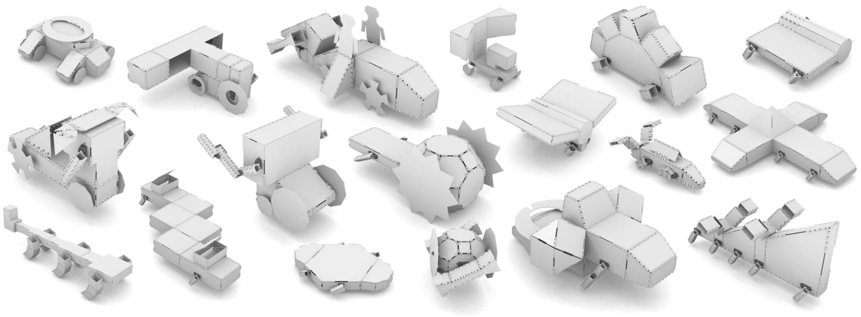
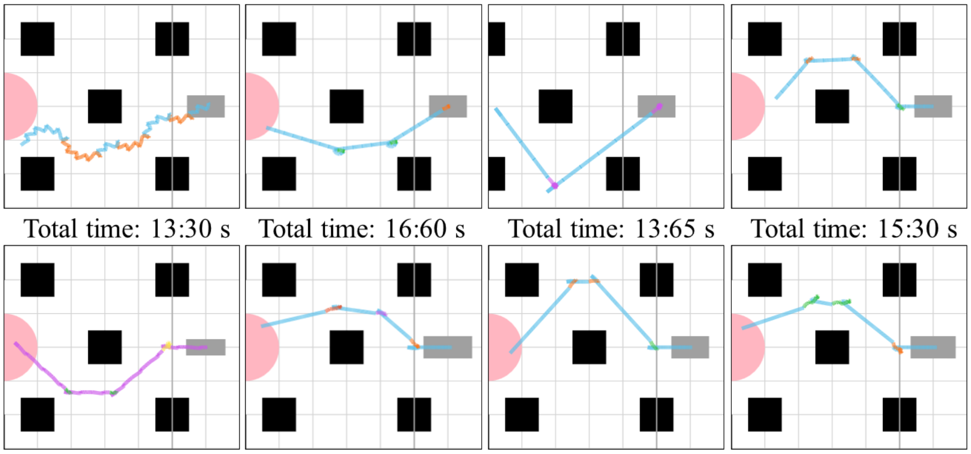


Figure 8-11: Gallery of designs created by a novice user after a 20 min. training session. Each of the models took between three and 25 min. to design and contains multiple components from the database.



Total time: 13:30 s Total time: 16:60 s Total time: 13:65 s Total time: 15:30 s

Total time: 9:25 s Total time: 8:10 s Total time: 9:30 s Total time: 8:40 s

Figure 8-12: Trajectories designed by users during the second task. Users who were allowed to make changes to the robot geometry were able to make the robot navigate the course about 40% faster on average.

to manipulate the geometry were able to explore the gait design tool creatively to reach the goal. Solutions included a combination of right and left rotation gaits (user 1), fixing the back legs and moving forward stably with only the front legs (users 2 and 6), and performing an 180 degree rotation before moving with a backwards gait towards the goal. To reach these solutions, users made extensive use of the interactive feedback during manipulation of gait parameters and gait sequence composition. The users reported that the most useful features were the animations and the trajectory visualization.

On the other hand, users who were not restricted to gait design edited the robot dimensions so that it would be stable during a forward gait with all legs moving simultaneously. The resulting gait sequences were significantly faster (40% time reduction on average) than the ones designed without geometry modification. These users referenced the metrics tab on the user interface when manipulating the shape to find more stable configurations. One user used the guide arrows to minimize the wobbliness metric.

Fabrication Examples We tested the full design pipeline by composing and fabricating six robot models, shown in Figure 8-13. Figure 8-14 shows the electronics inside one of the models. They each took 10-15 min. to design, 3-7 hr. to print, and 30-90 min. to assemble. Each robot was fabricated and assembled successfully and was able to execute the gait designed in the system.

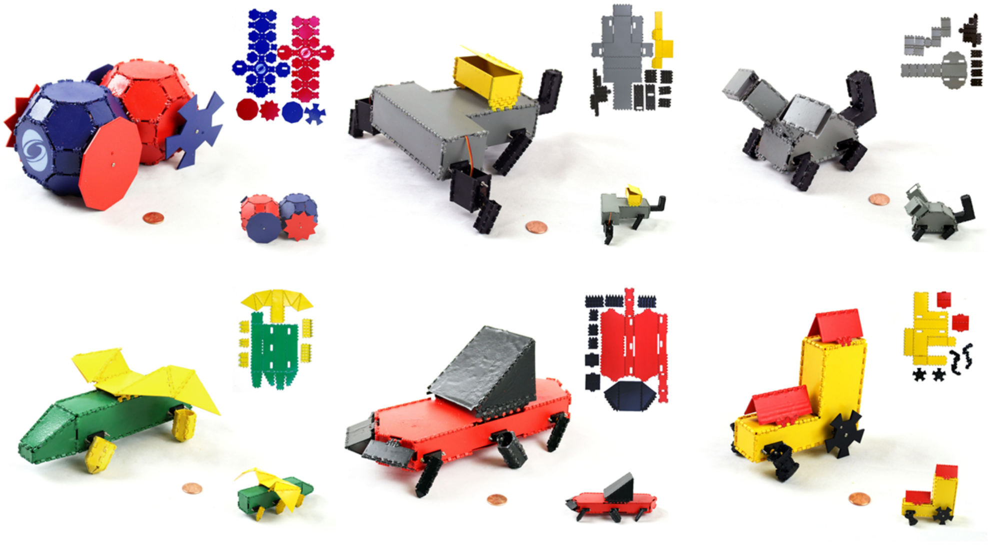


Figure 8-13: Six robots designed and fabricated using the system.

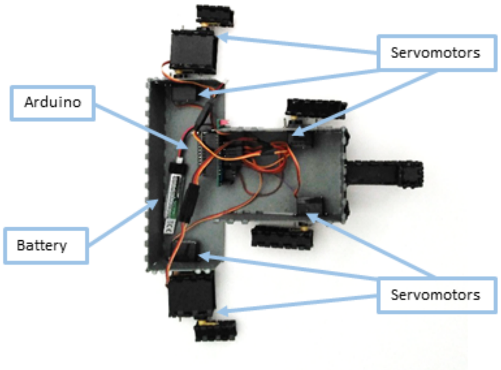


Figure 8-14: The electronics (servomotors, microcontroller, and battery) inside the Monkey example (Figure 8-13(d)).

The resulting robot trajectories were similar to those predicted by the system. Figure 8-15 shows frames from the simulation of the monkey compared with the physical robot at the same time. Each of the legs moves at the expected times and in the correct order. In addition, the robot dips forward when the third leg moves ($t = 3.0$ s) in both the simulation and in the physical model. Both models have turned slightly to the right by the end of one gait cycle.

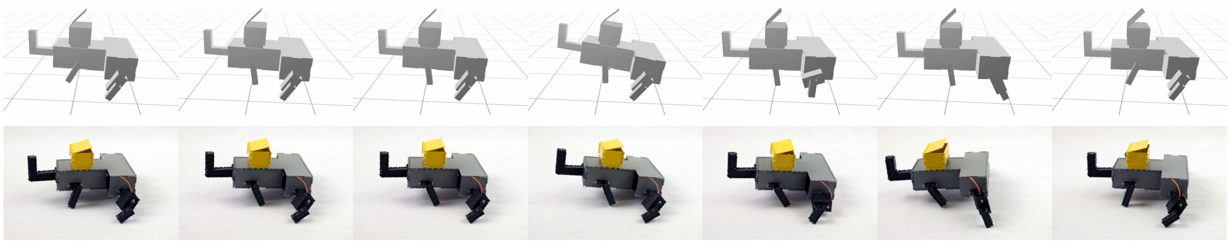


Figure 8-15: Comparison of monkey gait in simulation and from physical robot. The robot motion matches those from the simulations.

8.3 Interactive Multicopter Design

Multicopters are aerial vehicles that are mechanically simple and can be controlled manually or automatically to have a stable and accurate motion. While these vehicles are increasingly being used in many settings—including photography, disaster response, search and rescue operations, hazard mitigation, and geographical 3D mappings—most current multicopter designs are fairly standard (e.g., symmetric quadcopters or hexacopters).

Designing a non-standard multicopter that is optimized for a specific application is challenging since it requires expert knowledge in aerodynamics and control theory. However, we argue that expanding this design space can help create objects that are more adequate for some specific tasks. We therefore propose a design process that allows non-expert users to build custom multicopters that are optimized for specific design goals.

8.3.1 System Overview

Figure 8-16 shows the overview of our system. Users specify the geometry of the multicopter with our composition-based design tool. This tool allows the design of a parametric multicopter with geometric constraints on shape parameters and manufacturability guarantees. Our system automatically computes an LQR controller [Bouabdallah et al., 2004, Hoffmann et al., 2004] for the model.

Next, the user can optimize the vehicle by selecting a performance metric from a list of options or defining a weighted combination of these metrics. The optimization algorithm optimizes over both the shape and controller and returns results within seconds. A flight simulation is shown to allow users to validate the optimization results and tweak models appropriately. Once the user is satisfied with the design, the system outputs a complete fabrication plan for the geometry and controller. The users can then build and fly the designs.

8.3.2 Methods Overview

The central idea in this work in to parametrize both the geometry and the controllers. The geometry parametrization is done in the same way as the ground robots—following the ideas discussed in Chapter 3 and using composition rules for assembly-based modeling (see Fig-

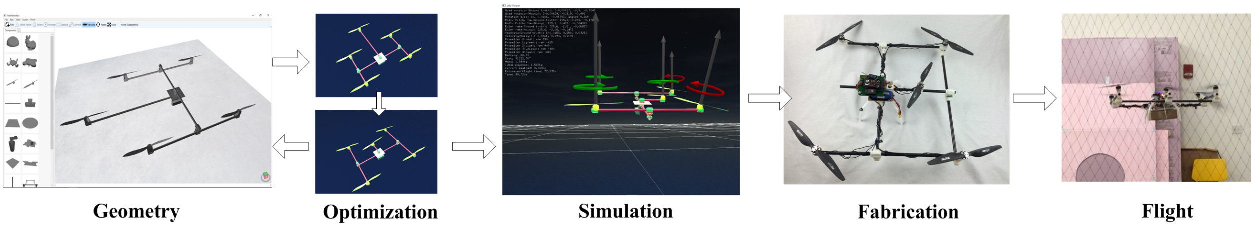


Figure 8-16: An overview of our system. The pipeline consists of multicopter design, optimization, simulation, fabrication, and flight tests.

ure 8-17). The parametric component database includes standard parts and parameterized free-form body frames.

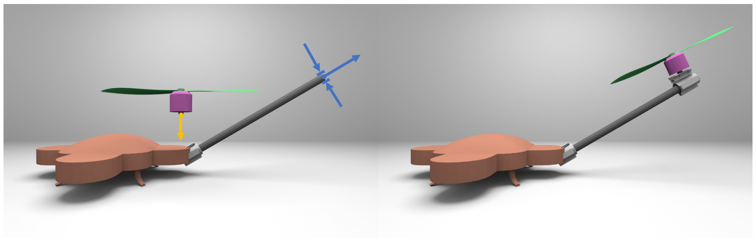


Figure 8-17: Example of composition. Left: parts with highlighted patches (circular patch with an annotated main axis and a diameter in blue and flat patch with an annotated normal in orange). Right: composed design.

The controller is designed by linearizing the state-space model at a fixed point and then finding a controller for that fixed point. We define the fixed point as the parameters of the controller that we will use to optimize the multicopter for flying capability. We define the following metrics over which to optimize the shape: cost, mass, payload, max amperage, size, and flight time. Since we used a linear model for parametrization of the geometry, we can show that these metrics can be expressed as bi-convex functions of the geometry and control parameters. This allows us to solve the optimization problem in seconds (see [Du et al., 2016]).

8.3.3 Results

Pentacopter optimized for payload Figure 8-18 shows a multicopter with five rotors all pointing upright that was designed in our tool. Our system can automatically compute the controller for this model.

Given the initial unoptimized pentacopter, our optimization improves its payload by changing its geometry and tilting motors to balance thrust from all motors. Figure 8-19 shows the real-time output of all five motors when this pentacopter carries maximal payload from one place to another.

Figure 8-19 compares the real-time output of all five motors when both pentacopters carry maximal payload from one place to another. Table 8.1 compares their specifications. Our optimization result predicts the new pentacopter is able to take off with a 15.8% increase

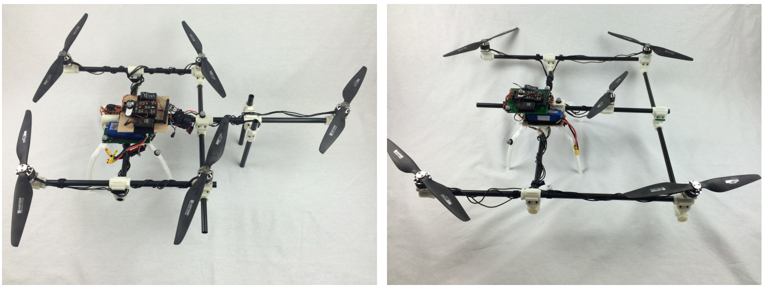


Figure 8-18: Pentacopter pairs. Left: original pentacopter design. Right: optimized pentacopter for larger payload.

in the overall weight. Note that in theory the maximal possible increase in the overall weight is less than 25% because even the unoptimized pentacopter outperforms a quadcopter whose maximal overall weight is $4u^{\max}$, and the maximal possible overall weight of a pentacopter is not greater than $5u^{\max}$. Table 8.1 shows that we get 11.1% actual gain in our experiments. The main reason for this loss is that we did not take into account the interference between propellers, which we leave as future work.

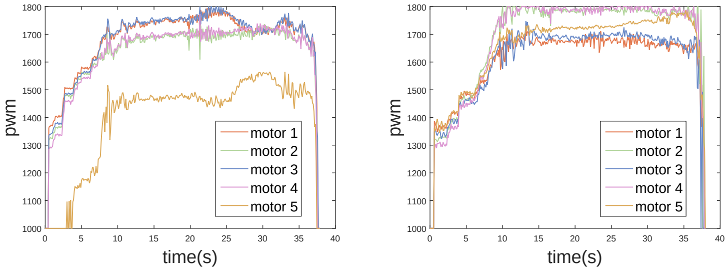


Figure 8-19: Left: motor outputs of the unoptimized pentacopter with 1047g payload. Motor 1 and 3 reach saturation point (PWM=1800) at 23s. Increasing the payload will cause them to saturate constantly and therefore fail to balance the torques from other motors. Note that motor 5 is not fully exploited in this copter. Right: motor outputs of the optimized pentacopter with 1392g payload. Motor 2 and 4 reach saturation during the flight. Compared with the unoptimized pentacopter, all five motors are now well balanced, making it possible to take over 30% more payload.

Quadcopter optimized for flight time Figure 8-20 shows a standard quadcopter and its optimized version which has longer flight time. Our optimization tries to increase the total length of rods so that it makes room for larger propellers. In this example we add an upper bound constraint on the copter width so it only scales in the other direction, allowing us to replace the propellers in the longer rod with larger ones. This geometry constraint is useful when a quadcopter is designed to fly into a tunnel. Both copters are controlled by LQR controllers computed with our system. In our experiments, we let both copters hover for 5 minutes and record the battery voltage and current, shown in Figure 8-21.

	Unoptimized	Optimized
Size (mm×mm×mm)	750×420×210	650×670×210
Weight (g)	2322	2353
Max payload (g)	1047	1392
Max overall weight (g)	3369	3745
Max motor angle (degree)	0	10.6

Table 8.1: Pentacopter specifications. Motor angle is the angle between motor orientation and up direction.

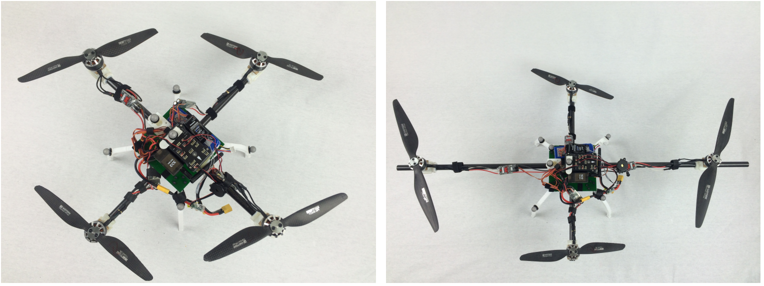


Figure 8-20: Optimizing a quadcopter with flight time metric and geometry constraints. Left: a standard quadcopter. Right: optimized rectangular quadcopter.

8.4 Robot-Assisted Carpentry

Despite the ubiquity of carpentered items, the customization of carpentered items remains labor intensive. The generation of laymen editable parametric databases for carpentry is difficult. Current design tools rely heavily on CNC fabrication, limiting applicability. We develop a design system for carpentry and a robotic fabrication system using mobile robots and standard carpentry tools. Our end-to-end design and fabrication tool democratizes design and fabrication of carpentered items. Our method combines expert knowledge for parametric design, allows laymen users to customize and verify specific designs, and uses a robotic system to fabricate parts.

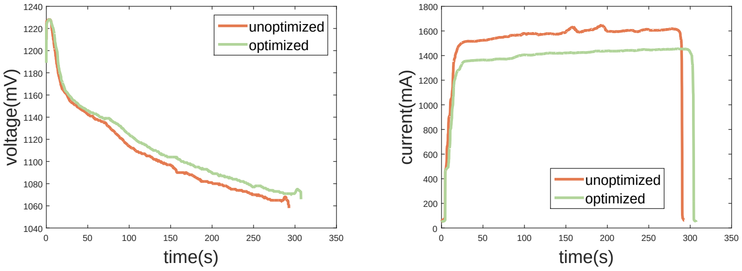


Figure 8-21: Battery change when a quadcopter hovers. Left: battery voltage. Given the same amount of time the optimized quadcopter ends up having a larger voltage. Right: battery current. In steady state the optimized copter requires less current.

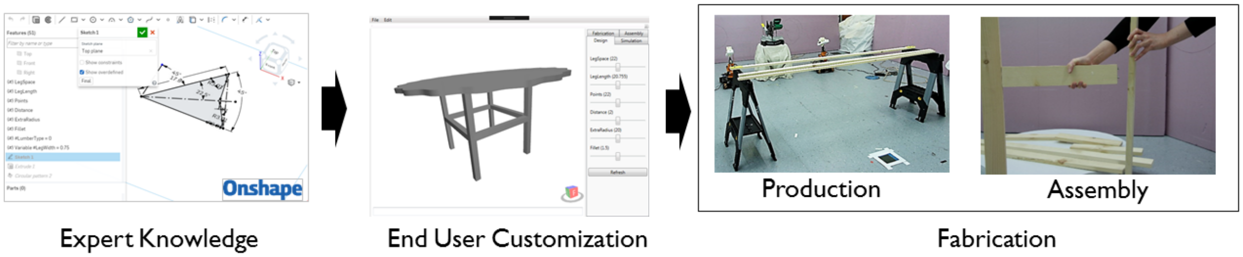


Figure 8-22: System workflow. Experts design parametric models in a commercial CAD system. End users customize and verify the designs using our interactive interface. Once the users are satisfied with the model, the system outputs a complete fabrication plan which includes instructions for robot assisted cutting processes and rules for user assembly.

8.4.1 Systems Overview

Our workflow allows collaborations between expert designers and layman users. The experts are responsible for creating designs with a set of exposed parameters which define different possible configurations for the models. The end users can vary these parameters to customize the parametric models to meet desired specifications and then fabricate the results assisted by a team of robots (see Figure 8-22).

When engineers design they should consider the available fabrication processes. In our system, experts are instructed to design parametric models based on a pre-specified set of carpentry tools. We choose to use a standard CAD software for modeling so engineers do not have to learn new tools to use our system. CAD systems are parametric from construction, allowing the engineer to expose a set of parameters with ranges to define a feasible design space for the layman user [Farin et al., 2002]. The engineer then annotates the connections between parts with priority tags. Our algorithm uses these tags to automatically compute assembly instructions.

We expose a simple interface for the end users where the designs can be customized by varying the parameters using sliders (see Figure 8-22). In addition to exploring the parameter space, our interface allows the user to visualize the stress distribution and deformation under certain loading conditions. This enables verification of designs before fabrication.

Once the users are satisfied with a design configuration, the system outputs a complete fabrication plan. The fabrication plan includes cut patterns and assembly instructions. We use two robot assisted cutting processes: a robotic team using chop saws, and mobile robotic jigsaws. After the robots finish cutting all the parts, the users assemble them guided by an interactive interface with assembly instructions.

8.4.2 Methods Overview

We use CAD systems to define parametric models that manufactured as a set of parts that can be cut with the set of predefined processes and then assembled. By exposing a set of parameters and selecting valid ranges, expert users specify variations to part shape that preserve structural integrity and manufacturability (see top row of Figure 8-23). In addition to the set of parts, the expert users also need to specify the connectors in order. To speed up this process, we used dowel peg connectors and defined a custom CAD feature that takes in two parts and a connecting face and automatically outputs a set of pegs. The custom feature also takes in priority tags. By adding these features to the design for all connections (see Figure 8-23(bottom)) the expert automatically defines the connection and assembly. Finally, the expert defines the boundary conditions for the simulation using a custom feature we designed. The feature takes in a face and the type of boundary condition: fixed boundary, or a boundary with an incident force to a given direction (see Figure 8-23(middle)).

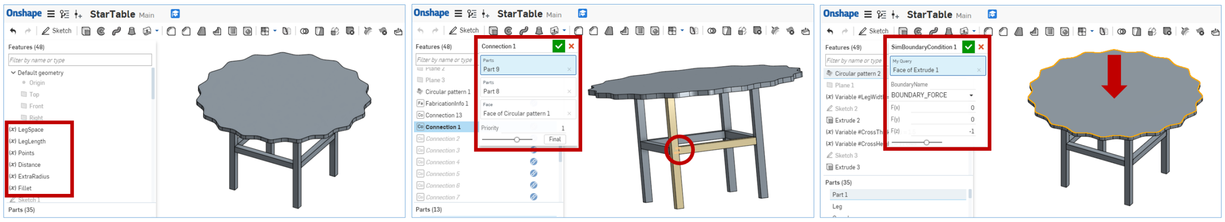

Onshape
StarTable Main
Features (48)
LegSpace
LegLength
Points
Distance
ExtraRadius
Fillet
Parts (35)
Connection 1
Part 9
Part 8
Face of Circular pattern 1
Priority
Features (49)
SimBoundaryCondition 1
Face of Extrude 1
BoundaryName
BOUNDARY_FORCE

Figure 8-23: Expert input using CAD software. Experts create a model in a standard CAD system which is parametric from construction and exposes the parameters for user customization using the system's variables (top). For defining the assembly, experts use a custom *connection* feature which takes in two parts, a face for connection and the priority of the connection for assembly (bottom). The experts use this feature to define all of the connections on the model and those are used by our algorithm to automatically generate assembly instructions.

Users can customize the templates by exploring the parameter space. We use the CAD system's API to evaluate and display the model for each parameter configuration. We also extract the parametric boundary conditions and run FEM analysis on the model and display to the users the stress distribution and elastic deformation (see Figure 8-24).

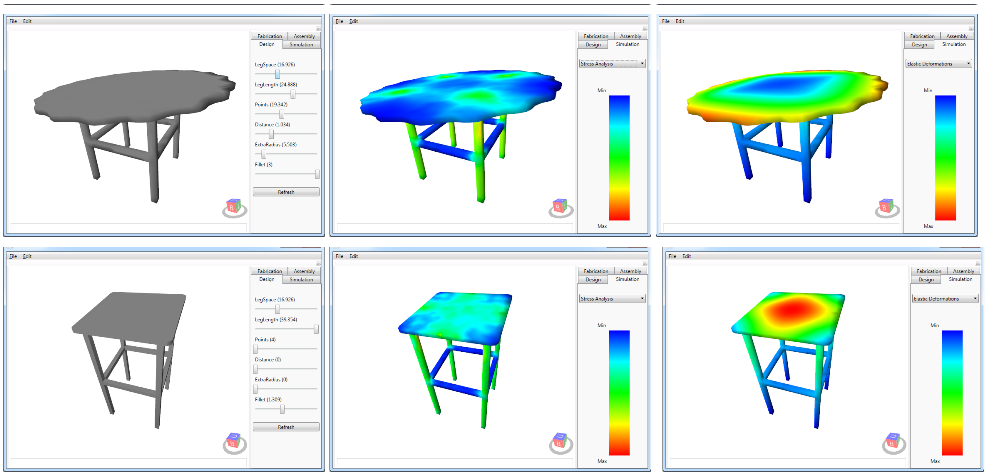

Fabrication
Assembly
Design
Simulation
LegSpace
LegLength
Points
Distance
ExtraRadius
Fillet
Refresh
Stress Analysis
Elastic Deformations
Min
Max

Figure 8-24: User customization: the design tab (top) and the stimulation tab that display the stress distribution (middle) and elastic deformations (bottom).

We use two fabrication processes: chop saw and jigsaw, both automated by robotic systems. This jigsaw robot is a modified Roomba Create with a jigsaw installed in the center. It uses a Vicon positioning system for state estimation and a previously developed MPC and planning algorithm to perform the cuts [Lipton et al., 2017]. The chop saw process requires multiple robots to automate. Two Kuka Youbots lift lumber and place it on the chop saw. Each Youbot was outfitted with special compliant grippers. The grippers allow the robots to clamp onto material, to drive the material along a direction, and are compliant perpendicular to the major axis of the lumber. The chop saw is automated by attaching a relay to the 120V line and a linear actuator is attached to the saw. Both are connected to a ROS node via a micro-controller (see Figure 8-25 and [Lipton et al., 2018] for details).

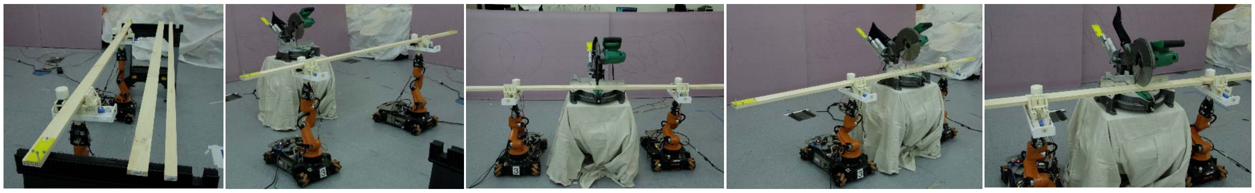
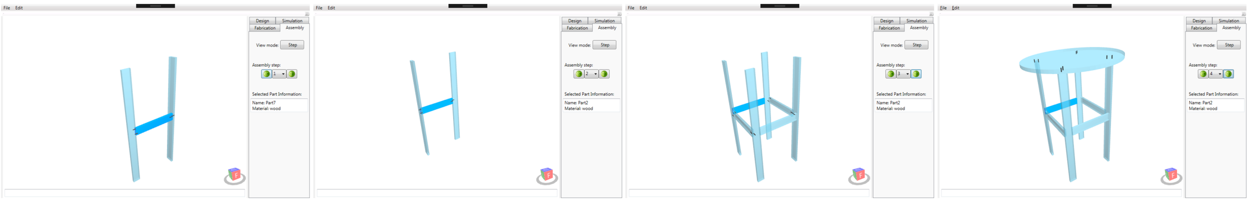


Figure 8-25: Chop saw process from left to right: Robots team lift the lumber, transport it, place it on the chop saw to re-grasp, slide the lumber to the proper length, and the chop saw cuts the lumber.

In order to guide the layman user through the process of assembly, we automatically generate a visual guide based on the parts and the set of connections for a given model instance (see Figure 8-26).



View mode: Step
Assembly step: 1
Selected Part Information:
Name: Part7
Material: wood

View mode: Step
Assembly step: 2
Selected Part Information:
Name: Part2
Material: wood

View mode: Step
Assembly step: 3
Selected Part Information:
Name: Part2
Material: wood

View mode: Step
Assembly step: 4
Selected Part Information:
Name: Part2
Material: wood

Figure 8-26: Sequence of assembly steps shown in the composition interface. The user can traverse the list of steps, select parts to view additional information, and use the 3D window to view the connections from different viewpoints.

8.4.3 Results

We show in Figure 8-27 a wide variety of geometry variation that resulted from customization of single parametric models. The figure displays the stress distribution for the chair model and elastic deformation for the shed model for different parameter configurations. As shown in this figure, the physical properties of the models vary significantly with geometry changes. Using our tools users can quickly explore this space of variations and verify that the designs meet the necessary physical requirements. While the variations allow for a diversity of models, the ranges imposed by the engineer limit the space so that all variations are structure preserving.

Our method automatically generated a fabrication plan for each of these variations, with results shown in Table 8.2. The table displays the number of parts that need to be

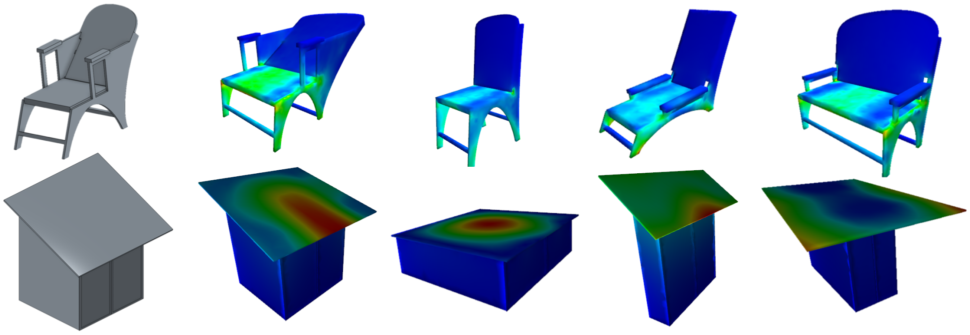


Figure 8-27: Stress distribution on different variations of the chair model (top) and elastic deformation on variations on the shed model (bottom).

manufactured with each of the cut processes (jigsaw and chop saw), the total number of pegs used, and the number of assembly steps. Our method is robust to discrete variations of the shape. In the chair example, a parameter determines the presence or absence of armrests. When these are absent the assembly requires one less step. In the shed example, we see how variations in size affect the number of pegs and variation in the number of back slabs affects the number of chop saw parts.

Table 8.2: Fabrication information for the models in Figure 8-27: number of parts that will be processed with the jig saw and chop saw, total number of used pegs, and number of assembly steps.

	Jigsaw Parts	Chop Saw Parts	Pegs	Assembly Steps
Chair A	4	10	86	5
Chair B	4	8	62	4
Chair C	4	10	66	5
Chair D	4	10	114	5
Shed A	16	37	1050	9
Shed B	16	25	563	9
Shed C	16	25	1333	9
Shed D	16	25	1121	9

One of the main applications for customization is the need to adapt to the surrounding environment. We show how our system can be adapted to terrain using the deck model shown in Figure 8-28. In this example, the terrain acts as a parameter in the system and the parametric deck is designed to accommodate the terrain variations. Figure 8-28 shows the input terrains and how the shape and physical properties of the deck change with the terrain variations.

As a test of the end to end system, an instance of the table design was made. The table design has eight components that need to be fabricated using the chop saw, and one using the jigsaw. The fully fabricated parts can be seen in Figure 8-29. Holes for the pegs were added in a manual step. A human user followed the step-by-step instructions from the user interface to assemble the table. The use of pegs, and tool-less nature of the assembly along

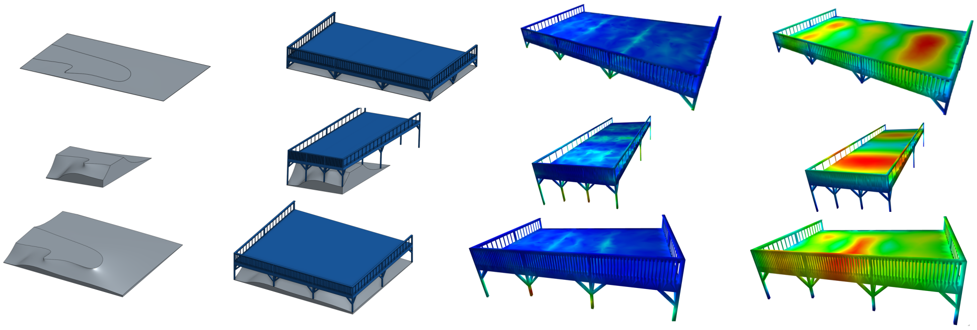


Figure 8-28: Variations of the deck model. From left to right: input terrain, deck model instance visualized on the terrain, stress distribution, elastic deformation.

with the user instructions provided a simple assembly experience.

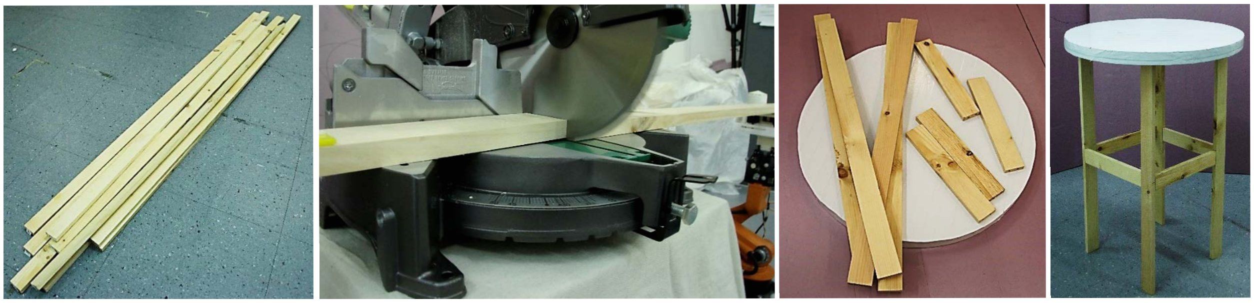


Figure 8-29: Fabrication of the table model. From left to right: eight pieces of stock 1x3 lumber, the cutting of the pieces, the final parts, and the assembled table result.

Chapter 9

Conclusion

In this work, we proposed the first complete data-driven design tool for manufacturing. In doing so, we created and released a dataset for computational fabrication, defined a new algorithm for shape-based matching and retrieval on a dataset of parametric designs, and defined a novel algorithm for composition that ensures manufacturability.

We also defined new performance-driven computational methods that guide users in efficiently exploring design spaces while optimizing for given performance objectives. We proposed a method for interactive design space exploration, which is based on a novel interpolation algorithm on k-d trees that guarantees locality and continuity. We also proposed a method for interactive exploration of performance trade-offs, which is built upon a first-order approximation of the Pareto front derived from duality theory in multi-objective optimization.

Finally, we presented end-to-end systems for design and manufacturing that combine data-driven and performance-driven approaches. We described a design tool for robots with ground locomotion that can handle concurrent specification of motion and geometry, as well as a design tool for multicopters that can handle concurrent optimization of geometry and control. We also presented a method for customization of carpentered items that can be fabricated with mobile robots.

9.1 Future Work

The work described in this thesis demonstrates how to create innovative and powerful manufacturing design tools that use data-driven methods to reduce the design space to manufacturable regions and then employ performance-driven methods to find optimal designs in these reduced spaces. However, our results so far are only first steps toward applying these concepts to computational design for manufacturing. Since the advances in manufacturing hardware are quite recent, the field of developing the necessary design tools is still relatively new, with many interesting challenges and open research problems. Below we outline several important open problems that we think are crucial to pushing the field forward.

9.1.1 Data-Driven Methods

With such a large and complex design space, using datasets to extract expert knowledge is essential to reduce the search space in interactive tools. Some of the main avenues for future work in the context of data-driven methods are: 1) creating and maintaining databases, 2) extracting additional properties besides manufacturability from existing datasets, and 3) defining algorithms that are not limited by the dataset and can allow for greater design diversity.

Data Collections Large, publicly-available design collections with complete manufacturing detail do not currently exist. In this thesis, we have worked with mechanical engineering students to build a database of manufacturable models that we have made public. In the future, it will be valuable to extend these efforts and work on creating, keeping, and growing such databases. This would involve building a strong community of engineers and designers, along with verification systems that would allow inputs to the databases to be efficiently crowd-sourced. The developed datasets will further promote the field of computational methods for manufacturing, as well as provide evaluation benchmarks for new algorithms and systems. Most importantly, as advanced manufacturing platforms become ubiquitous, they will become an invaluable resource for the entire population, enabling people from all walks of life to control the creation and customization process of their possessions.

Design Capabilities In this work, we discussed how databases can be used to constrain design spaces based on manufacturing considerations. In the future, it will be exciting to use data-driven methods to enable other design capabilities.

First, we could use data-driven methods to predict physical behavior. Extending the database to include usage data such as the typical forces an object encounters in the course of use would allow us to more rigorously validate a design's durability. Such information, in aggregate, can also be used to learn efficient ways to prioritize between objectives to meet higher level goals, or even how to suggest new performance metrics. Additionally, because the items in our database are continually reused, there are opportunities to leverage preprocessing to dramatically accelerate our physics computations.

Second, we can learn from data collections how to predict or simplify the design process. For example, it would be interesting to present relevant components to the user during the modeling session. Chaudhuri et al. [2011] propose a probabilistic model that suggests components based on style and geometrical semantics. In the context of fabrication, such suggestions should also include metrics such as construction time, required tools, and currently available materials.

Third, we can use this dataset and composition method to automatically synthesize new fabricable models by finding plausible combinations of components, as accomplished for virtual shapes by Kalogerakis et al. [2012]. An interesting opportunity is to use ideas from the Program Synthesis community to design automatic programming algorithms for inverse design of geometry, which can be represented as code using CAD-inspired abstractions. Next, we can extend these ideas for functional design of complex mechanisms that require inverse design on multiple domains by defining domain-specific languages based on the parametric abstractions—for example, the grammar for ground robots discussed in Section 8.2.

Finally, we can output assembly instructions along with the bill of materials. Though the output of our system could be used as input to the automatic algorithm proposed by Agrawala et al. [2003], it would be interesting in the future to add assembly information to the catalog and use this data to automatically generate instructions. A relevant open problem for outputting valid assembly instructions is ensuring that the relevant parts of the shape can be accessed with the necessary hand tools in the construction process.

Result Diversification One of the main limitations of data-driven methods is that design variations are constrained by expert specifications. In the tools we have developed so far, the only significant shape variety that can be achieved is driven by composition. An important avenue of future work is to define solutions for discovering designs that significantly differ from the dataset beyond modular combinations of designs. It would be interesting to learn design rules from the data collection and generalize them to discover solutions that lie outside the scope of the dataset and are currently unknown to experts.

A possible approach is to learn engineering design *bias* from data collections. Bias is common in engineering design because engineers tend to replicate styles they have seen work in practice, because the engineering education is typically based on systematic methodologies, and because CAD systems encourage a certain style of designing. Large datasets of designs made with the same engineering intent can be used to extract common features. We can then expose bias from these features by analyzing how they affect performance. Understanding this bias can drive design algorithms that optimize performance by suggesting feature edits that are uncommon and perhaps unintuitive to engineers.

9.1.2 Performance-Driven Methods

In this work, we show efficient ways to map between design and performance spaces. Future work on performance-driven methods can further exploit this mapping to extract high-level information of design capabilities and analyze how design parameters affect performance. We can also extend the discussed ideas to handle more complex domains and more complex objectives.

Design Space Analysis The mapping onto performance space reveals important aspects of the design space that can be analyzed to give engineers a better understanding

of the design space's capabilities and limitations. For example, it can be used to expose design features that are more (or less) relevant for performance optimization, or to highlight the extent to which performance metrics are conflicting, allowing engineers to more effectively select between possible trade-offs. An interesting result of our first-order approximation algorithm (Chapter 7) that should be further explored is how variations in design space can locally affect performance trade-offs. Such information on the design behavior on the boundary of the performance space can be used to discover interesting directions of expansion in design space, defining ways in which a model can deviate from expert specifications to improve performance.

Extensions to Complex Domains An important limitation of the performance-driven methods we discuss in this work is the restriction to continuous domains. In the future, it would be useful to develop techniques for interactive exploration and optimization of design spaces that combine discrete and continuous parameters. It would also be interesting to develop algorithms to handle high dimensional problems. In this work, we proposed interactive design exploration techniques that handle high dimensional performance spaces but are restricted to low dimensional design spaces (Chapter 6). On the other hand, the algorithms for interactive exploration of performance trade-offs is subject to the curse of dimensionality in performance space, but can handle high dimensional design spaces (Chapter 7). In the future, it would be useful to investigate solutions that can handle problems where both spaces are high dimensional.

Handling Complex Objectives Future work should also consider design problems that involve more complex performance objectives.

First, it would be interesting to handle performance objectives that cannot be evaluated computationally. While physical simulation is available in many practical design applications, this is not always the case. Consider for example the design of a solar panel, where the thickness of each composite material is a design parameter. In such example, there is no simulation method that can accurately compare the performance of each design (e.g. closed current amperage). An alternative to simulation is to physically produce and measure samples of solar panels and then approximate the mapping function from design space to performance using machine learning algorithms. In such context, an important technical problem is defining a strategy that minimizes the number of needed samples. This would be done by developing new sampling strategies based on the application domain—for example, a strategy for finding the Pareto frontier with the minimal number of samples. Another possibility is to incorporate priors derived from physics rules.

Second, in many practical applications, designers care about higher-level performance metrics that are not easily mapped to physical measurements. For example, the design of a shoe is typically optimized for “comfort” and not a specific deformation function under external forces. A related open question is aesthetics. While aesthetic objectives are hard to express as performance metrics, they are in practice an important aspect of design that is typically considered and optimized in the design process. Mapping from high-level objectives to low-level objectives that can be measured directly is an interesting avenue for future work.

Third, it is important to handle uncertainty when evaluating performance objectives. Uncertainty is common not only because of inaccuracies in simulation, but also because for many manufacturing processes the physical realizations deviate from the design specifications due to fabrication errors. Such errors have been shown to highly impact design performance Kim et al. [2017].

Finally, performance objectives may depend on design parameters. Consider, for example, a bench with varying thickness. As the thickness increases, the number of people that can sit on the bench varies, and, consequently, the boundary conditions for a deformation analysis also vary. In the future, it would be useful to investigate coupled design and performance spaces.

9.1.3 End-to-End Systems

We argue that building end-to-end systems is a fundamental aspect of research in computational design for manufacturing not only because they allow proposed methodologies to impact real-world applications, but also because they expose fundamental technical challenges in the field. Some important aspects of system design that need further investigation are: 1) hardware abstractions, 2) domain-specific systems, and 3) interfacing with and between users.

Hardware Abstractions One of the big open problems in end-to-end computational design systems for manufacturing is to define proper abstractions for the diverse array of manufacturing hardware that is becoming available (3D printers, CNC, knitting machines, laser cutters, robots, etc.). Hardware abstractions have been successfully applied to computers, abstracting machine language into assembly languages and then to high-level languages. This allows programmers to write high-performance applications that are device-independent. On the other hand, most of the manufacturing hardware that exists today allows only for a very rudimentary level of control, similar to assembly-level instructions. To advance computational design it is fundamental to develop guidelines for defining domain-specific languages for single devices and to then extend those guidelines to a set of devices that operate together. We can take advantage of the resulting compact and dense representation of search spaces for inverse design problems. On the compiler front, we can use these abstractions to optimize fabrication through scheduling and execution of parallel processes.

Domain-Specific Systems Another important avenue for future work is to expand these systems to other domains. It would be interesting to handle more complex, integrated cyber-physical systems. This would require incorporating dynamic simulations and verification, as well as environmental or task constraints such as load, dynamic forces, and robustness. In designing systems that are accessible to users with less expertise, however, being *domain-specific* is useful since expanding the capabilities of a tool also increases the learning curve. In this work, we have developed different tools for different problems (e.g. ground robots and multicopters). Ultimately, it would be interesting to allow automatic generation of domain-specific computational tools—a tool for designing design tools.

Interfacing with Users From a human-computer interaction perspective, there are two key problems to address. The first is to understand ways to interface between a single user and the computation. In this work, we use designer expertise either directly or through data to constrain design spaces and define performance objectives. We then develop computational tools to automatically explore design and performance. However, as both of these spaces increase in dimensionality and complexity, we will need tools that allow smooth collaborations between human and computation to iteratively constrain and define these spaces. This will allow us to find solutions that are better than what a designer alone or a computer alone would find.

The second important aspect is to allow collaborations between users. One important question is how designers with different areas of expertise can co-design an object with multiple functionality. Another interesting area to explore is collaboration between experts and casual designers. The systems we discuss in this thesis are based on a linear workflow: experts create a dataset that casual designers then explore for customization and composition. In the future, closed-loop collaborations can be developed to improve experts' understanding of end-users needs, which is fundamental for human-centered design.

9.2 Lessons Learned

Throughout the course of this thesis and beyond the technical contributions, we have learned a great deal about the design process and important considerations that affect design tools for manufacturing.

The first important lesson is the need to develop complete end-to-end systems and use them to build physical prototypes that can validate design algorithms and techniques. Developing and having users test these systems is essential to understanding the capabilities and limitations of the proposed approaches and to provide insight into fundamental design problems.

Second, we have learned that there is a need to balance between automation and user input. In addition to the well-known conflict between ease of use and design freedom, such balance is important to develop efficient design tools. This is because there are tasks where computers tend to outperform humans and vice-versa. Multiple times in this work, we have found that offloading small tasks to users helped us define algorithms that achieved better results in less time.

Third, we have learned that good representations are key. Having compact representations of shape spaces and relationships between assembled parts allowed us to define simpler algorithms that can run in real-time. One important example of this is the use of parametric shapes, which allowed us to express many complex operations (e.g. snapping components) as an optimization over a small set of parameters that can be solved at interactive rates.

Finally, we have learned that it is essential to interface with different research communities. The work discussed in this thesis draws ideas from a variety of different fields, and therefore lies at the intersection of computer graphics, geometry processing, data analytics, programming languages, robotics, mechanical engineering, and human-computer interaction.

Appendix A

Proofs of Interpolation Algorithm

Our approach takes advantage of how the centers of odd B-splines at different levels are distributed across the domain. In this supplemental material, we first describe this distribution making the necessary notations, then prove properties of the basis refinement step. Finally, we use the above results to prove the local point lemma and that locality is preserved when we define y as in Section 5.2 to ensure linear precision. We conclude this supplemental material with an example that illustrates the effects of the proposed refinement method in the resulting approximation function.

A.1 Notation

As discussed in Section 5.2, we denote as c_i^j the center of the linear B-splines ϕ_i^j. Let $\mathcal{I}^j$ be the lattice of centers c_i^j of linear B-splines at the refinement level j. Observe that, if $t > j$, then $\mathcal{I}^j \subset \mathcal{I}^t$. Crucially, linear B-splines at refinement level j can be centered only at the lattice points $\mathcal{I}^j$; at each successive level of refinement, this lattice becomes twice as fine (i.e., the distance between adjacent points in the lattice is halved). This is illustrated in Figure A-1.

The size of the support of a linear B-spline at level j is denoted by $\mathbf{s}_j$ (Figure A-1). We let $\|\cdot\|_d$ denote the length of an element in direction d.

Using the expression in Section 5.3, we say that ϕ_i^j is active if $\exists k$ such that $\alpha_k^{i,j} \neq 0$.

Using the refinement relations, we say that a B-spline ϕ_n^{j+1} is a child of ϕ_i^j (equivalently, that ϕ_i^j is a parent of ϕ_n^{j+1}) if ϕ_n^{j+1} results from the refinement of ϕ_i^j, i.e., if the refinement coefficient a_{in}^{j+1} (Equation 4) is not zero. From Figure 6, it is clear that one function can have multiple children and multiple parents and a function will have a single parent if and only if they have the same center.

A.2 Properties of Step 2

Remark 1. *For every active linear B-spline ϕ_i^j:*

$$\mathbf{s}_j \leq 2\|e_l\|_d \quad \forall e_l, e_l \cup S(\phi_i^j) \neq \emptyset. \tag{A.1}$$

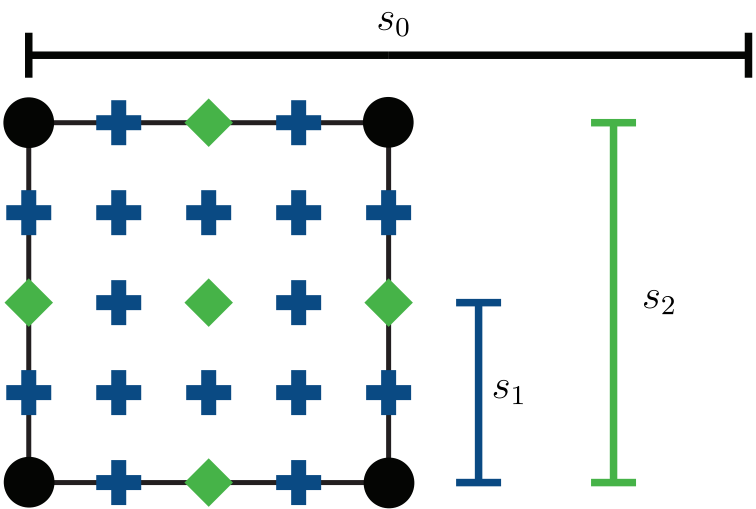


Figure A-1: Centers of linear B-splines at different levels. The set $\mathcal{I}^0$ consists only of the centers denoted by circles, the set $\mathcal{I}^1$ includes the centers denoted by circles and those marked by diamonds, and $\mathcal{I}^2$ includes all the centers denoted in the figure.

Proof. In the initial configuration, defined in Section 4.2, we have a single element e_0 and a set of linear B-splines ϕ_i^0 at the coarsest level centered at corners of this element such that $\mathbf{s}_0 = 2\|e_0\|_d$ in all directions d.

In each iteration, when an element e_l is split, we refine all linear B-splines ϕ_i^j which overlap that split element and have $\mathbf{s}_j > \|e_l\|_d$. Therefore the above statement follows from an inductive argument. ☐

Remark 2. *If ϕ_i^j is refined, then it has no active ancestors*

Proof. In step 2, a linear B-spline ϕ_i^j is refined if $\mathbf{s}_j > \|e_l\|_d$. If it had an active parent, ϕ_n^{j-1}, then its support would be twice as large and therefore $\mathbf{s}_{j-1} > 2\|e_l\|_d$, violating the property proved in Remark 1. Any previous ancestor would have an even larger support, which concludes the proof. ☐

These remarks will be used in the following proofs of locality and we can also conclude from them that at each iteration step 2 needs to perform at most one level of refinement.

A.3 Local Point Lemma

In this section we will prove that if c_i^j is the center of an active linear B-spline ϕ_i^j, then c_i^j is a local point of ϕ_i^j.

Remark 3. *$S(\phi_i^j)$ overlaps with at most two elements in a given direction and if it overlap with two elements then c_i^j is at the boundary between these elements.*

Proof. Since the locations of the centers of the linear B-splines c_i^j are fixed and given by $\mathcal{I}^j$, it is clear from Figure A-1 that ϕ_i^j that overlaps e_l will overlap another element e_k if and only if e_l and e_k are adjacent and the center c_i^j lies on the boundary between e_l and e_k.

Since element refinement involves splitting an element halfway and basis functions are refined to guarantee $\|e_l\|_d \leq 2\mathbf{s}_j$ for all active ϕ_i^j which overlaps e_l, this property is preserved after every iteration of our algorithm. □

This remark implies that if $S(\phi_i^j)$ overlaps with e_l, then $e_l \in \mathcal{N}(c_i^j)$. From this we conclude that $S(\phi_i^j) \subset \mathcal{N}(c_i^j)$, and therefore c_i^j is a local point of ϕ_i^j from the definition (recall Section 4).

Since we have shown that the center c_i^j is a local point, this concludes the proof of the local point lemma.

A.4 Locality proof with Linear Precision

If ϕ_i^j is active, let y_i^j be, as defined as in Section 5.2,

$$y_i^j = \sum_k \alpha_k^{i,j} x_k / \alpha_i^j, \quad \text{where} \quad \alpha_i^j = \sum_k \alpha_k^{i,j} \tag{A.2}$$

In this section we will show that if ϕ_i^j is active, then then y_i^j is a local point of ϕ_i^j.

We will use an inductive argument. From the proof given in the previous section, in the initial configuration when $y_i^0 = c_i^0$, y_i^0 is a local point. We will show that this property is preserved at every iteration of our algorithm, i.e., it is preserved when basis functions are refined when elements are split.

A.4.1 Preservation over Basis Refinement

When a B-spline ϕ_i^j is refined it becomes inactive and the the coefficients $\alpha_k^{j+1,n}$ of its children ϕ_n^{j+1} are updated. This results in updating the positions y_n^{j+1} to $\bar{y}_n^{j+1}$. To show that the above property is preserved over basis refinement, it is sufficient to prove that $\bar{y}_n^{j+1}$ is a local point of ϕ_n^{j+1}.

Let us first consider the case when the ϕ_n^{j+1} was inactive before refinement of ϕ_i^j. Let a_{in}^{j+1} be the refinement coefficient given by Equation 4. In this scenario, the refinement of ϕ_i^j updates the coefficients $\alpha_k^{n,j+1}$ as follows (Equation 8):

$$\bar{\alpha}_k^{n,j+1} = a_{in}^{j+1} \alpha_k^{i,j}, \qquad \forall k. \tag{A.3}$$

From the definition of y (Equation A.2) that

$$\bar{y}_n^{j+1} = \frac{\cancel{a_{in}^{j+1}} \sum_k \alpha_k^{i,j} x_k}{\cancel{a_{in}^{j+1}} \sum_k \alpha_k^{i,j}} = y_i^j. \tag{A.4}$$

From the refinement property $S(\phi_n^{j+1}) \subset S(\phi_i^j)$, the induction assumption $S(\phi_i^j) \subset \mathcal{N}(y_i^j)$, and the above result $\bar{y}_n^{j+1} = y_i^j$ it follows that $S(\phi_n^{j+1}) \subset \mathcal{N}(\bar{y}_n^{j+1})$. Therfore $\bar{y}_n^{j+1}$ is a local point of ϕ_n^{j+1}.

Let us now consider the case when ϕ_n^{j+1} was active before refinement of ϕ_i^j. We make the flowing remarks:

Remark 4. *If ϕ_n^{j+1} is an active child of ϕ_i^j, then the updated position of y_n^{j+1}, given by $\bar{y}_n^{j+1}$, can be expressed as a convex combination of y_i^j and y_n^{j+1}.*

Proof. Let a_{in}^{j+1} be the refinement coefficient given by Equation 4. The refinement of ϕ_i^j updates the coefficients $\alpha_k^{n,j+1}$ as follows (Equation 8):

$$\bar{\alpha}_k^{n,j+1} = \alpha_k^{n,j+1} + a_{in}^{j+1}\alpha_k^{i,j}, \qquad \forall k. \tag{A.5}$$

Therefore, if y is defined as in Equation A.2, then

$$\bar{y}_n^{j+1} = \frac{\alpha_n^{j+1} y_n^{j+1} + a_{in}^{j+1}\alpha_i^j y_i^j}{\alpha_n^{j+1} + a_{in}^{j+1}\alpha_i^j}. \tag{A.6}$$

□

Remark 5. *If ϕ_n^{j+1} is active, $\mathcal{N}(y_i^j) \cap \mathcal{N}(y_n^{j+1}) \subset \mathcal{N}(\bar{y}_n^{j+1})$*

Proof. If $e_l \in \mathcal{N}(y_i^j) \cap \mathcal{N}(y_n^{j+1})$, then $y_i^j, y_n^{j+1} \in e_l$. Since $\bar{y}_n^{j+1}$ is a convex combination of y_i^j and y_n^{j+1}, $\bar{y}_n^{j+1} \in e_l$ and therefore $e_l \in \mathcal{N}(\bar{y}_n^{j+1})$. □

From the induction assumption $S(\phi_n^{j+1}) \subset \mathcal{N}(y_n^{j+1})$ and $S(\phi_i^j) \subset \mathcal{N}(y_i^j))$. Since ϕ_n^{j+1} results from the refinement of ϕ_i^j, $S(\phi_n^{j+1}) \subset S(\phi_i^j)$ and therefore $S(\phi_n^{j+1}) \subset \mathcal{N}(y_i^j) \cap \mathcal{N}(y_n^{j+1})$. From Remark 5 we conclude that $S(\phi_n^{j+1}) \subset \mathcal{N}(\bar{y}_n^{j+1})$, showing that the property is preserved during basis refinement.

A.4.2 Preservation over Element Refinement

Finally we will show that $S(\phi_i^j) \subset \mathcal{N}(y_i^j)$ is preserved when an element are split. We start by making the following remarks.

Remark 6. *Let ϕ_i^j be an active linear B-spline, $j > 0$. If ϕ_n^{j-1} are the parents of ϕ_i^j which have been refined, then*

$$\alpha_i^j = \sum_n a_{ni}^j$$

$$y_i^j = \sum_n a_{ni}^j c_n^{j-1} / (\sum_n a_{ni}^j) \tag{A.7}$$

where the coefficients a are given by Equation 4.

Before we prove this Remark, we prove the following Remark that stems directly from it.

Remark 7. *If ϕ_i^j is active and has no active ancestors, then $\alpha_i^j = 1$ and $y_i^j = c_i^j$.*

Proof. For $j = 0$, this results directly from the initial configuration when all linear B-splines are at the coarsest level and $c_i^0 = y_i^0$ and $\alpha_i^0 = 1$.

For $j > 0$ we will use the result from Remark 6:

On the one dimensional case for linear B-splines, using the values of a from Equation 5, we can write Equation A.7 when all parents are refined as

$$\begin{cases} y_{2i}^j = c_i^{j-1} \\ y_{2i+1}^j = \frac{1}{2} c_i^{j-1} + \frac{1}{2} c_{i+1}^{j-1}. \end{cases}$$

From the lattice structure and the symmetry of the a terms around the center, we see that $y_i^j = c_i^j$ when all parents are refined. From Equation 5, $\alpha_i^j = \sum_n a_{ni}^j = 1$, which is a result from the partition of unity property of refinement relations. This result is directly extended in the multi-dimensional case.

□

Proof. [Remark 6] We will prove this property by induction. At the initial configuration when all linear B-splines are at the coarsest level, $c_i^0 = y_i^0$ and $\alpha_i^0 = 1$. At this level, there are no active linear B-splines with $j > 0$ and therefore the Remark 6 holds.

We assume that Remark 6 holds and will show that after an iteration of the refinement algorithm it still holds.

We first show that it still holds after step 2. Let ϕ_i^j be a linear B-spline which whill be refined in this step. Only linear B-splines with no active ancestors can be refined (Remark 2) and therefore ϕ_i^j has no active ancestors. Since we assume that Remark 6 holds, Remark 7 also holds and therefore $y_i^j = c_i^j$ and $\alpha_i^j = 1$. Let ϕ_n^{j+1} be a child of ϕ_i^j.

If ϕ_n^{j+1} is inactive, then

$$\begin{aligned} \bar{\alpha}_n^{j+1} &= a_{in}^{j+1} \alpha_i^j = a_{in}^{j+1} && \text{(from Equation A.3)} \\ \bar{y}_n^{j+1} &= y_i^j = c_i^j && \text{(from Equation A.4)} \end{aligned} \tag{A.8}$$

and Remark 6 holds.

Otherwise, if ϕ_n^{j+1} is not inactive, then

$$\begin{aligned} \bar{\alpha}_n^{j+1} &= \alpha_n^{j+1} + a_{in}^{j+1} && \text{(from Equation A.5)} \\ \bar{y}_n^{j+1} &= \frac{\alpha_n^{j+1} y_n^{j+1} + a_{in}^{j+1} c_i^j}{\alpha_n^{j+1} + a_{in}^{j+1}} && \text{(from Equation A.6).} \end{aligned} \tag{A.9}$$

From the induction assumption $\alpha_n^{j+1} y_n^{j+1} = \sum_m a_{im}^{i+1} c_m^j$ and $\alpha_n^{j+1} = \sum_m a_{im}^i$, for m indexing all parents ϕ_m^{j+1} other than ϕ_n^{j+1} that have been refined. Therefore

$$\bar{\alpha}_n^1 = \sum_m a_{im}^i + a_{in}^{j+1}$$

$$\bar{y}_n^{j+1} = \frac{\sum_m a_{im}^{i+1} c_m^j + a_{in}^{j+1} c_t^j}{\sum_m a_{im}^i + a_{in}^{j+1}}$$

and Remark 6 holds. From this we conclude that the property is preserved after step 2.

In step 3, though the values $\alpha_k^{i,j}$ are updated, the values α_i^j do not change (Equation 10), concluding the proof.

$\square$

We will now show that $S(\phi_i^j) \subset \mathcal{N}(y_i^j)$ is preserved when e_l is split into e_{lA} and e_{lB} across direction d. It is sufficient to validate the statement on the active linear B-splines ϕ_i^j that overlap e_l. From the induction assumption y_i^j is a local point of e_l. We must then prove that if $S(\phi_i^j)$ overlaps with e_{lA} (and/or e_{lB}), then y_i^j is a local point of e_{lA} (and/or e_{lB}). We will proceed to prove this considering the two possible cases: $S(\phi_i^j)$ overlaps with only one element (e_{lA} or e_{lB}) and $S(\phi_i^j)$ overlaps with both elements (e_{lA} and e_{lB}).

Case 1: ϕ_i^j overlaps one element Consider a linear B-spline ϕ_i^j that overlaps with e_l. From the induction assumption ($S(\phi_i^j) \in \mathcal{N}(y_i^j)$) $y_i^j \in e_l$. First, let us consider the case when ϕ_i^j overlaps only one of the elements that result from the split. Without loss of generality, we assume $S(\phi_i^j) \cap e_{lB} = \emptyset$. If $y_i^j \notin e_{lA}$, then this element refinement would make y_i^j no longer a local point. In what follows we will show that this is never the case, i.e., if $S(\phi_i^j) \cap e_{lB} = \emptyset$, then $y_i^j \in e_{lA}$, from which we can conclude that $S(\phi_i^j) \in \mathcal{N}(y_i^j)$.

Remark 8. *If ϕ_i^j is active, then $y_i^j \in \overline{S(\phi_i^j)}$, where $\overline{S(\phi_i^j)}$ is the closure of $S(\phi_i^j)$.*

Proof. This property holds in the initial configuration when $y_i^0 = c_i^0$. We will show that this property is preserved during basis refinement (step 2) and conclude the proof by induction. As in the previous paragraph, we will look at the updated positions $\bar{y}_n^{j+1}$ when ϕ_n^{j+1} results from the refinement of ϕ_i^j.

From Remark 2 and Remark 7, if ϕ_i^j is refined, then $y_i^j = c_i^j$. The refinement relations guarantee that for linear B-splines $c_i^j \in \overline{S(\phi_n^{j+1})}$ (see Figure 6) from which we conclude $y_i^j \in \overline{S(\phi_n^{j+1})}$.

If y_n^{j+1} is not active $\bar{y}_n^{j+1} = y_i^j$ (Equation A.4) and therefore $\bar{y}_n^{j+1} \in \overline{S(\phi_n^{j+1})}$.

Otherwise, if y_n^{j+1} is active, $y_n^{j+1} \in \overline{S(\phi_n^{j+1})}$ from the induction assumption. Remark 4 allows us to express $\bar{y}_n^{j+1}$ and a convex combination of y_n^{j+1} and y_i^j (both in $\overline{S(\phi_n^{j+1})}$) from which we conclude that $\bar{y}_n^{j+1} \in \overline{S(\phi_n^{j+1})}$.

$\square$

From the induction assumptions ($y_i^j \in e_l$) and Remark 8, $y_i^j \in e_l \cap \overline{S(\phi_i^j)}$. From the assumption $S(\phi_i^j) \cap e_{lB} = \emptyset$, $\overline{S(\phi_i^j)} \cap e_{lB} \subset \partial e_{lB}$, where ∂e_{lB} is the boundary of e_{lB}. From this we conclude that $y_i^j \in e_{lA} \cup \partial e_{lB}$. Since $S(\phi_i^j)$ is a K-dimensional cuboid and e_{lA} and e_{lB} are adjacent K-dimensional cuboids we conclude from $S(\phi_i^j) \cup e_{lA} \neq \emptyset$ that $\overline{S(\phi_i^j)} \cup \partial e_{lb} \setminus e_{lA} = \emptyset$. Therefore, $y_i^j \in e_{lA}$ and therefore $S(\phi_i^j) \subset \mathcal{N}(y_i^j)$ after element refinement.

Case 2: ϕ_i^j overlaps both elements Now let us consider the case when ϕ_i^j overlaps both elements that result from the split, i.e., $S(\phi_i^j) \cap e_{lA} \neq \emptyset$ and $S(\phi_i^j) \cap e_{lB} \neq \emptyset$. As previously discussed, $y_i^j \in e_l$. In what follows we will prove that y_i^j is also on the boundary between e_{lA} and e_{lB}, which will allow us to conclude that after the split the property $S(\phi_i^j) \in \mathcal{N}(y_i^j)$ is preserved.

Let us first consider the case when there are no active ancestors and therefore $\sum_k \alpha_k^{i,j} = 1$ and $y_i^j = c_i^j$ (Remark 7). From Remark 3, if ϕ_i^j overlaps both elements, then c_i^j is on the boundary between e_{lA} and e_{lB} and therefore $S(\phi_i^j) \in \mathcal{N}(c_i^j)$. Since in this case $y_i^j = c_i^j$, the property is preserved.

Let us now consider the case when there are active ancestors. Let $\mathcal{B}_i^j$ be the set of active ancestors.

Remark 9. *$\mathcal{B}_i^j$ describes the set of active linear B-splines $\phi_n^m \neq \phi_i^j$ that do not vanish at c_i^j*

Proof. The B-splines $\phi_n^m \neq \phi_i^j$ that do not vanish at c_i^j are its ancestors (all of them) or descendents that are centered at c_i^j (see Figure 6). All of the active ancestors are in $\mathcal{B}_i^j$. Since ϕ_i^j is active, there can be no active linear B-spline whose only ancestor is ϕ_i^j. Since the descendents that do not vanish at c_i^j have ϕ_i^j as a unique ancestor, none of them are active. □

From Remark 9 and the assumption that a partition of unity is guaranteed in all previous iterations, we conclude

$$\sum_k \alpha_k^{i,j} + \sum_{n,m\in\mathcal{B}_i^j} \sum_k \alpha_k^{n,m} \phi_n^m(c_i^j) = 1. \tag{A.10}$$

If $S(\phi_i^j)$ overlaps e_{lA} and e_{lB} the same is true for all of its ancestors. Therefore, from Remark 3, they must all be centered on the boundary between e_{lA} and e_{lB} . We conclude that c_n^m is equal to c_i^j in direction d, $c_n^m|_d = c_i^j|_d$. We will use this to show that $y_i^j|_d = c_i^j|_d$.

Let m_0 be the coarsest level in $\mathcal{B}_i^j$. Any function $\phi_n^{m_0}$ must have no active ancestors since those would also be on $\mathcal{B}_i^j$. From Remark 7, $y_n^{m_0} = c_n^{m_0}$ and therefore, $y_n^{m_0}|_d = c_i^j|_d$.

Next, we take the next coarsest level on $\mathcal{B}_i^j$, m_1, and let $\phi_{n_1}^{m_1}$ be any linear B-spline at this level. $\mathcal{B}_{n_1}^{m_1}$ only contains the linear B-splines in $\mathcal{B}_i^j$ at level m_0. As in Equation A.10, we can use Remark 9 and the partition of unity assumption to conclude

$$\sum_k \alpha_k^{n_1,m_1} + \sum_n \sum_k \alpha_k^{m_0,n} \phi_n^{m_0}(c_{n_1}^{m_1}) = 1$$

Using the assumption of linear precision on all previous iterations and letting the evaluation function $x_k \mapsto p_k$ be the identity, we conclude (Equation 7) that

$$c_{n_1}^{m_1} = \sum_k \alpha_k^{n_1,m_1} x_k + \sum_n \sum_k \alpha_k^{m_0,n} x_k \phi_n^{m_0}(c_{n_1}^{m_1}).$$

From this we can express $c_{n_1}^{m_1}$ as a convex combination of $y_{n_1}^{m_1}$ and $y_n^{m_0}$.

$$c_{n_1}^{m_1} = \alpha_{n_1}^{m_1} y_{n_1}^{m_1} + \sum_n \phi_n^{m_0}(c_{n_1}^{m_1}) \alpha_{m_0}^n y_{m_0}^n.$$

Since $c_{n_1}^{m_1}|_d = y_n^{m_0}|_d = c_i^j|_d$, it follows that $y_{n_1}^{m_1}|_d = c_i^j|_d$. We can continue this process to the finer levels to achieve that $y_i^j|_d = c_i^j|_d$.

From this we conclude that y_i^j is a local point after an element is split which concludes the proof that y_i^j is a local point.

A.5 Example

Let $q_i^j = \sum \alpha_k^{i,j} p_k$, where the coefficients $\alpha_k^{i,j}$ are given by Equation 6. Then, Equation 7 can be expressed as:

$$P(x) = \sum_{i,j} q_i^j \phi_i^j(x). \tag{A.11}$$

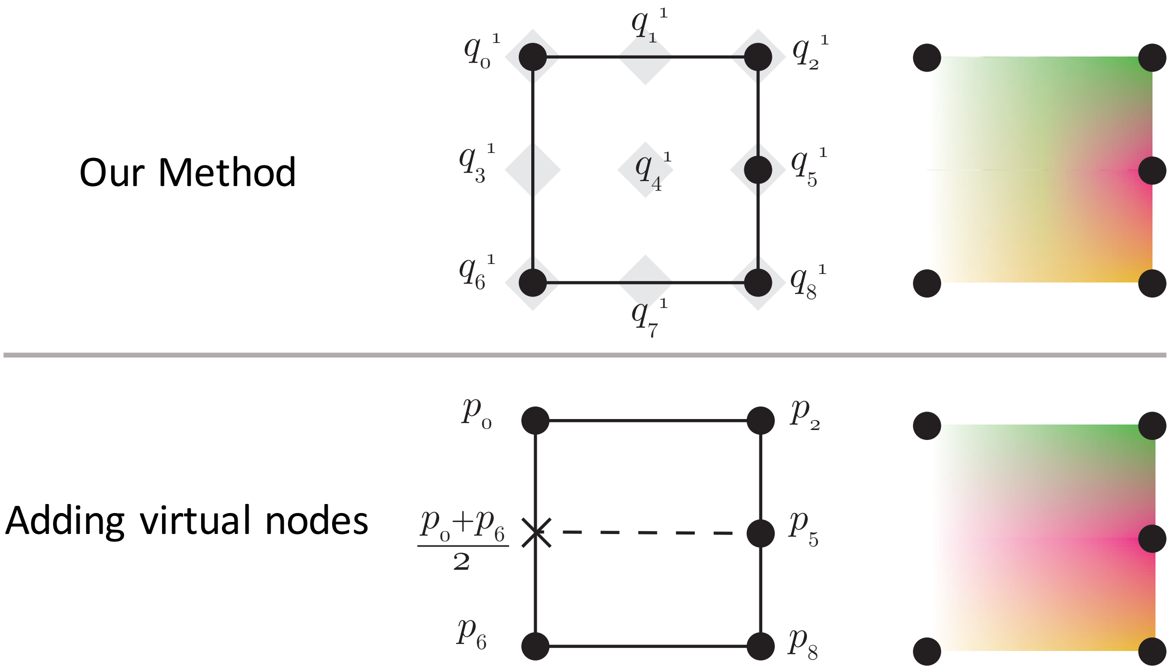


Figure A-2: Contrasting the interpolation solution from our method with the interpolation solution adding virtual nodes. Top row: the interpolation solution from our method. The resulting interpolation is equivalent to a uniform basis function at the coarsest level j such that every sample on the element boundary belongs to $\mathcal{I}^j$ (in this case $j = 1$). The values q_i^j can be computed hierarchically at points for which samples p_i^j do not exist. In this example, q_3^1 is based on the average of the adjacent samples $x_0, x_6 \in \mathcal{I}^0$, and q_4^1 is the average of the four corner samples, also contained in $\mathcal{I}^0$. Bottom row: interpolation solution adding virtual nodes. The color display on the right illustrates how our method restricts the impact of a sample in $\mathcal{I}^j$ to $\mathbf{s}_j$.

To further illustrate the result of our refinement strategy for high dimensions, we show a two-dimensional example on the top row of Figure A-2. Given an element and the samples x_k on its boundary, we can determine the solution of our approximation. We use Equation A.11 with uniform basis functions at the coarsest level j at which $x_k \in \mathcal{I}^j, \forall k$. The coefficients q_i^j can be determined at each successive level j as follows. At $j = 0$, since there are guaranteed to be samples at $x_k = x_i^0$, we set $q_i^0 = p_k$. At level $j > 0$, the coefficient q_j^i at a point $x_k = y_i^j$ for which a sample does not exist is given by a multi-linear combination of coefficients $q_i^{(j-1)}$ at the coarser level $j - 1$.

Figure A-2 compares our method (depicted in the top row) with the approach of creating virtual nodes and then using bilinear interpolation on each sub-element (shown in the bottom row). The figure highlights the difference in the effect of p_5. In our technique, if j is the coarsest level such that $x_k \in \mathcal{I}^j$, then the influence of the sample x_k is limited to the support of basis functions at level j. As we have shown in this supplemental material, this property is used to prove locality. Therefore, this property gives us the advantage of being able to define a simple refinement algorithm that updates the approximation while constructing a k-d tree.

Notice that the same result on the top row of Figure A-2 can be achieved by the following steps: first, use a set of basis functions at the coarsest level, setting $P(x) = \sum_{i=0,2,6,8} \phi_i^0 p_i$; then, use the refinement relations to rewrite this expression as $\sum_{i=0}^{8} \phi_i^1 q_i^1$; finally, replace q_5^1 in this expression, which was originally $(p_2 + p_8)/2$, with p_5. The second advantage of expressing this interpolation as basis functions with refinement relations is that this method can be extended to higher-order basis functions, such as cubic B-splines.

Appendix B

Proof of the First-Order Approximation of the Pareto Front

Since $\mathbf{x}(t) \in \mathcal{P}$ for all $t \in (-\varepsilon, \varepsilon)$, each point $\mathbf{x}(t)$ must satisfy the KKT conditions (7.2). Hence, we can assume the existence of time-varying dual variables $\alpha(t) : (-\varepsilon, \varepsilon) \to \mathbb{R}^d$ and $\beta(t) : (-\varepsilon, \varepsilon) \to \mathbb{R}^K$. Generically these functions are differentiable in t.

For a given critical point $\mathbf{x}^* = \mathbf{x}(0)$, without loss of generality permute the constraints so that the first K' inequality constraints are active, i.e. $g_k(\mathbf{x}^*) = 0$ for all $k \leq K'$, and that the remaining constraints are inactive, i.e. $g_k(\mathbf{x}^*) < 0$ for all $k > K'$.

Applying the complementary slackness condition in (7.2), we must have $\beta_k(0) = 0$ for all inactive constraints $g_k(\mathbf{x}^*)$. By continuity, if a constraint is inactive at $t = 0$ it must remain inactive in a nonempty open interval surrounding $t = 0$. After possibly restricting ε, we can assume $\beta_k(t) \equiv 0$ for all $k > K'$ and $t \in (-\varepsilon, \varepsilon)$.

Collecting our observations so far, we rewrite the KKT conditions (7.2) as follows:

$$\left\{ \begin{array}{l} \alpha_i(t) \geq 0 \;\; \forall i \in \{1, \ldots, d\}, t \in (-\varepsilon, \varepsilon) \\ \beta_j(t) \geq 0 \;\; \forall j \in \{1, \ldots, K'\}, t \in (-\varepsilon, \varepsilon) \\ \beta_j g_j(\mathbf{x}(t)) = 0 \;\; \forall j \in \{1, \ldots, k\}, t \in (-\varepsilon, \varepsilon) \\ \sum_{i=1}^d \alpha_i(t) = 1, t \in (-\varepsilon, \varepsilon) \\ \sum_{i=1}^d \alpha_i(t) \nabla f_i(\mathbf{x}(t)) + \sum_{j=1}^{K'} \beta_j(t) \nabla g_j(\mathbf{x}(t)) = 0 \;\; \forall t \in (-\varepsilon, \varepsilon) \end{array} \right\} \tag{B.1}$$

Note this form effectively ignores the inactive constraints since they do not figure into the problem near $t = 0$.

Our next task is to differentiate the final condition in (B.1) with respect to t at $t = 0$. Define:

$$h(t) := \sum_{i=1}^d \alpha_i(t) \nabla f_i(\mathbf{x}(t)) + \sum_{j=1}^{K'} \beta_j(t) \nabla g_j(\mathbf{x}(t))$$

Then,

$$\begin{aligned} h'(t) &= DF^\top(\mathbf{x}(t))\alpha'(t) + \left(\sum_{i=1}^{d} \alpha_i(t) H_{f_i}(\mathbf{x}(t))\right)\mathbf{x}'(t) \\ &+ DG^\top(\mathbf{x}(t))\beta'(t) + \left(\sum_{k=1}^{K'} \beta_k(t) H_{g_k}(\mathbf{x}(t))\right)\mathbf{x}'(t) \end{aligned}$$

Here, Du indicates the Jacobian and H_u indicates the Hessian of a function $u(\mathbf{x})$.

Evaluating at $t = 0$ and recalling $\mathbf{x}(0) = \mathbf{x}^*$ shows

$$\begin{aligned} h'(0) &= DF^\top(\mathbf{x}^*)\alpha'(0) + \left(\sum_{i=1}^{d} \alpha_i(0) H_{f_i}(\mathbf{x}^*)\right)\mathbf{x}'(0) \\ &+ DG_{K'}^\top(\mathbf{x}^*)\beta'(0) + \left(\sum_{k=1}^{K'} \beta_k(0) H_{g_k}(\mathbf{x}^*)\right)\mathbf{x}'(0) \end{aligned}$$

We use $DG_{K'}$ to denote the part of the Jacobian of G corresponding to active constraints. Defining $\mathrm{H} := \sum \alpha_i(0) H_{f_i}(\mathbf{x}^*) + \sum \beta_k(0) H_{g_k}(\mathbf{x}^*)$ allows us to simplify our expression to

$$h'(0) = DF^\top(x^*)\alpha'(0) + DG_{K'}^\top(x^*)\beta'(0) + \mathrm{H}\mathbf{x}'(0). \tag{B.2}$$

From the KKT conditions, we know $h(t) \equiv 0$—and hence $h'(t) \equiv 0$—for all $t \in (-\varepsilon, \varepsilon)$. Rearranging slightly shows

$$H\mathbf{x}'(0) \in \mathrm{Im}(DF^\top(\mathbf{x}^*)) \oplus \mathrm{Im}(DG_{K'}^\top(\mathbf{x}^*)). \tag{B.3}$$

We obtain an additional property of $\mathbf{x}'(0)$ by revisiting the complementary slackness condition in (B.1), which shows $\beta_k(t) g_k(\mathbf{x}(t)) \equiv 0$ for $t \in (-\varepsilon, \varepsilon)$ and $k \in \{1, \ldots, K'\}$. Again differentiating both sides with respect to t shows

$$0 = \beta_k'(t) g_k(\mathbf{x}(t)) + \beta_k(t) \nabla g_k(\mathbf{x}(t))^\top \mathbf{x}'(t).$$

Recall $g_k(\mathbf{x}^*) = 0$ since constraint k is active; we furthermore can assume $\beta_k(0) \neq 0$ since the constraint is active. Hence the first term vanishes and at $t = 0$ we are left with

$$\nabla g_k(\mathbf{x}^*)^\top \mathbf{x}'(0) = 0 \ \forall k \in \{1, \ldots, K'\}, \tag{B.4}$$

Combining different k's shows

$$DG_{K'}(\mathbf{x}^*)\mathbf{x}'(0) = 0,$$

as desired.

Bibliography

Aseem Agarwala. Efficient gradient-domain compositing using quadtrees. In *Siggraph 2007*. ACM, 2007.

Shailen Agrawal and Michiel van de Panne. Pareto optimal control for natural and supernatural motions. 2013.

Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics*, 22(3):828–837, 2003.

Noam Aigerman, Roi Poranne, and Yaron Lipman. Lifted bijections for low distortion surface mappings. *ACM Trans. Graph.*, 33(4):69, 2014.

Noam Aigerman, Roi Poranne, and Yaron Lipman. Seamless surface mappings. *ACM Trans. Graph.*, 34(4):72:1–72:13, July 2015a. ISSN 0730-0301.

Noam Aigerman, Roi Poranne, and Yaron Lipman. Seamless surface mappings. *ACM Trans. on Graph. (TOG)*, 34(4):72, 2015b.

Marc Alexa, Daniel Cohen-Or, and David Levin. As-rigid-as-possible shape interpolation. In *Siggraph 2000*, pages 157–164. ACM, 2000.

Pierre Alliez and Craig Gotsman. Recent advances in compression of 3d meshes. In *Advances in multiresolution for geometric modelling*, pages 3–26. Springer, 2005.

Mihael Ankerst, Gabi KastenmÃijller, Hans-Peter Kriegel, and Thomas Seidl. Nearest neighbor classification in 3d protein databases. *In Proc. ISMB*, pages 34–43, 1999.

Melinos Averkiou, Vladimir Kim, Youyi Zheng, and Niloy J. Mitra. Shapesynth: Parameterizing model collections for coupled shape exploration and synthesis. *Computer Graphics Forum (Special issue of Eurographics 2014)*, 2014.

Ed Ayyappa. Normal human locomotion, part 1: Basic concepts and terminology. *Journal of Prosthetics and Orthotics*, 9(1):10–17, 1997.

Mehdi Baba-Ali, David Marcheix, and Xavier Skapin. A method to improve matching process by shape characteristics in parametric systems. *Computer-Aided Design and Applications*, 6(3):341–350, 2009.

Moritz Bächer, Stelian Coros, and Bernhard Thomaszewski. Linkedit: Interactive linkage editing using symbolic kinematics. *ACM Trans. Graph.*, 34(4):99:1–99:8, July 2015. ISSN 0730-0301.

Sunith Bandaru and Kalyanmoy Deb. Temporal innovization: Evolution of design principles using multi-objective optimization. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 9018 of *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, pages 79–93. Springer Verlag, 2015. ISBN 9783319159331. doi: 10.1007/978-3-319-15934-8_6.

Fan Bao, Dong-Ming Yan, Niloy J. Mitra, and Peter Wonka. Generating and exploring good building layouts. *ACM Trans. Graph.*, 32(4):122:1–122:10, July 2013. ISSN 0730-0301.

Ilya Baran. Onshape inc. Personal Communication, 2017.

David Benson and Joel Davis. Octree textures. *ACM Transactions on Graphics*, 21(3): 785–790, 2002.

Nicola Bezzo, Peter Gebhard, Insup Lee, Matthew Piccoli, Vijay Kumar, and Mark Yim. Rapid co-design of electro-mechanical specifications for robotic systems. In *ASME 2015 International Design Engineering Technical Conferences and Computers and Information in Engineering Conference*, pages V009T07A009–V009T07A009. American Society of Mechanical Engineers, 2015.

Gaurav Bharaj, David I. W. Levin, James Tompkin, Yun Fei, Hanspeter Pfister, Wojciech Matusik, and Changxi Zheng. Computational design of metallophone contact sounds. *ACM Trans. Graph.*, 34(6):223:1–223:13, October 2015. ISSN 0730-0301.

Bernd Bickel, Moritz Bächer, Miguel A. Otaduy, Hyunho Richard Lee, Hanspeter Pfister, Markus Gross, and Wojciech Matusik. Design and fabrication of materials with desired deformation behavior. *ACM Trans. Graph.*, 29(4):63:1–63:10, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778800. URL `http://doi.acm.org/10.1145/1778765.1778800`.

Rafael Bidarra and Willem F Bronsvoort. Semantic feature modelling. *Computer-Aided Design*, 32(3):201–225, 2000.

Rafael Bidarra, Paulos J Nyirenda, and Willem F Bronsvoort. A feature-based solution to the persistent naming problem. *Computer-Aided Design and Applications*, 2(1-4):517–526, 2005.

Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2006. ISBN 0387310738.

Martin Bokeloh, Michael Wand, Hans-Peter Seidel, , and Vladlen Koltun. An algebraic model for parameterized shape editing. *ACM Transactions on Graphics*, 31(4), 2012a.

Martin Bokeloh, Michael Wand, Hans-Peter Seidel, and Vladlen Koltun. An algebraic model for parameterized shape editing. *ACM Trans. Graph.*, 31(4):78:1–78:10, July 2012b. ISSN 0730-0301.

Samir Bouabdallah, Andre Noth, and Roland Siegwart. PID vs LQ control techniques applied to an indoor micro quadrotor. In *Intelligent Robots and Systems (IROS) 2004*, 2004.

Y. Boykov, O. Veksler, and R. Zabih. Fast approximate energy minimization via graph cuts. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 23(11):1222–1239, 2001.

Alexander M. Bronstein, Michael M. Bronstein, Leonidas J. Guibas, and Maks Ovsjanikov. Shape google: Geometric words and expressions for invariant shape retrieval. *ACM Trans. Graph.*, 30(1):1:1–1:20, 2011.

Siddhartha Chaudhuri and Vladlen Koltun. Data-driven suggestions for creativity support in 3d modeling. *ACM Transactions on Graphics*, 29(6):183:1–183:10, 2010.

Siddhartha Chaudhuri, Evangelos Kalogerakis, Leonidas J. Guibas, and Vladlen Koltun. Probabilistic reasoning for assembly-based 3d modeling. *ACM Transactions on Graphics*, 30(4):35, 2011.

Ding-Yun Chen, Xiao-Pei Tian, Yu-Te Shen, and Ming Ouhyoung. On visual similarity based 3d model retrieval. *Computer Graphics Forum*, 22(3):223–232, 2003.

Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3sweep: Extracting editable objects from a single photo. *ACM Trans. Graph.*, 32(6):195:1–195:10, 2013.

Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W. Sumner, Wojciech Matusik, and Bernd Bickel. Computational design of mechanical characters. *ACM Trans. Graph.*, 32(4):83:1–83:12, July 2013. ISSN 0730-0301. doi: 10.1145/2461912.2461953. URL `http://doi.acm.org/10.1145/2461912.2461953`.

David W Currier. Automation of sheet metal design and manufacturing. In *17th Conference on Design Automation*, pages 134–138. IEEE, 1980.

Indraneel Das and J. E. Dennis. Normal-boundary intersection: A new method for generating the pareto surface in nonlinear multicriteria optimization problems. *SIAM Journal on Optimization*, 8(3):631–657, 1998.

Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Proceedings of the twentieth annual symposium on Computational geometry*, pages 253–262. ACM, 2004.

K. Deb, L. Thiele, M. Laumanns, and E. Zitzler. Scalable Multi-Objective Optimization Test Problems. In *Congress on Evolutionary Computation (CEC 2002)*, pages 825–830. IEEE Press, 2002.

Kalyanmoy Deb and Kalyanmoy Deb. Multi-objective Optimization. In *Search Methodologies*, pages 403–449. Springer US, Boston, MA, 2014. doi: 10.1007/978-1-4614-6940-7_15. URL `http://link.springer.com/10.1007/978-1-4614-6940-7{_}15`.

Kalyanmoy Deb and Aravind Srinivasan. Innovization: Innovating design principles through optimization. In *Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation*, GECCO '06, pages 1629–1636, New York, NY, USA, 2006. ACM. ISBN 1-59593-186-4. doi: 10.1145/1143997.1144266. URL `http://doi.acm.org/10.1145/1143997.1144266`.

Manfredo Perdigao Do Carmo. *Differential geometry of curves and surfaces*, volume 2. Prentice-hall Englewood Cliffs, 1976.

Yue Dong, Jiaping Wang, Fabio Pellacini, Xin Tong, and Baining Guo. Fabricating spatially-varying subsurface scattering. *ACM Trans. Graph.*, 29(4):62:1–62:10, July 2010. ISSN 0730-0301. doi: 10.1145/1778765.1778799. URL `http://doi.acm.org/10.1145/1778765.1778799`.

Tao Du, Adriana Schulz, Bo Zhu, Bernd Bickel, and Wojciech Matusik. Computational multicopter design. *ACM Transactions on Graphics*, 35(6), 2016.

Gerald E Farin, Josef Hoschek, and Myung-Soo Kim. *Handbook of computer aided geometric design*. Elsevier, 2002.

Michael S Floater. Mean value coordinates. *Computer aided geometric design*, 20(1):19–27, 2003.

Michael S Floater. Generalized barycentric coordinates and applications. *Acta Numerica*, 24:161–214, 2015.

David R. Forsey and Richard H. Bartels. Hierarchical b-spline refinement. In *Siggraph 1988*, pages 205–212. ACM, 1988. ISBN 0-89791-275-6.

Michael Foshey, Nicholas Bandiera, and Javier Ramos. Mechanical engineers at mit. Personal Communication, 2017.

Thomas A. Funkhouser, Michael M. Kazhdan, Philip Shilane, Patrick Min, William Kiefer, Ayellet Tal, Szymon Rusinkiewicz, and David P. Dobkin. Modeling by example. *ACM Transactions on Graphics*, 23(3):652–663, 2004.

Ran Gal, Ariel Shamir, and Daniel Cohen-Or. Pose oblivious shape signature. *IEEE Transactions of Visualization and Computer Graphics*, 13(2):261âĂŞ–271, 2007.

Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. Iwires: an analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics*, 28(3), 2009.

Akash Garg, Andrew O. Sageman-Furnas, Bailin Deng, Yonghao Yue, Eitan Grinspun, Mark Pauly, and Max Wardetzky. Wire mesh design. *ACM Trans. Graph.*, 33(4):66:1–66:12, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601106. URL `http://doi.acm.org/10.1145/2601097.2601106`.

Zoubin Ghahramani, Geoffrey E Hinton, et al. The em algorithm for mixtures of factor analyzers. Technical report, Technical Report CRG-TR-96-1, University of Toronto, 1996.

Eitan Grinspun, Petr Krysl, and Peter Schröder. Charms: A simple framework for adaptive simulation. *ACM Trans. Graph.*, 21(3):281–290, July 2002. ISSN 0730-0301.

Satyandra K Gupta and Dana S Nau. Systematic approach to analysing the manufacturability of machined parts. *Computer-Aided Design*, 27(5):323–342, 1995.

Claus Hillermeier. *Nonlinear multiobjective optimization: a generalized homotopy approach*, volume 135. Springer Science & Business Media, 2001.

Gabe Hoffmann, Dev Gorur Rajnarayan, Steven L Waslander, David Dostal, Jung Soon Jang, and Claire J Tomlin. The Stanford testbed of autonomous rotorcraft for multi agent control (STARMAC). In *Digital Avionics Systems Conference (DASC) 2004*, 2004.

Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.*, 30(4):78, 2011.

Arjun Jain, Thorsten Thormählen, Tobias Ritschel, and Hans-Peter Seidel. Exploring shape variations by 3d-model decomposition and part-based recombination. *Comp. Graph. Forum (Proc. Eurographics 2012)*, 31(2), 2012.

Tao Ju, Scott Schaefer, and Joe Warren. Mean value coordinates for closed triangular meshes. In *ACM Transactions on Graphics (TOG)*, volume 24, pages 561–566. ACM, 2005.

Evangelos Kalogerakis, Siddhartha Chaudhuri, Daphne Koller, and Vladlen Koltun. A probabilistic model for component-based shape synthesis. *ACM Transactions on Graphics*, 31 (4), 2012.

Rahul Khardekar, Greg Burton, and Sara McMains. Finding feasible mold parting directions using graphics hardware. *Computer-Aided Design*, 38(4):327–341, 2006.

Jeeeun Kim, Anhong Guo, Tom Yeh, Scott E Hudson, and Jennifer Mankoff. Understanding uncertainty in measurement and accommodating its impact in 3d modeling and printing. In *Proceedings of the 2017 Conference on Designing Interactive Systems*, pages 1067–1078. ACM, 2017.

Vladimir G. Kim, Wilmot Li, Niloy J. Mitra, Siddhartha Chaudhuri, Stephen DiVerdi, and Thomas Funkhouser. Learning part-based templates from large collections of 3d shapes. *ACM Transactions on Graphics*, 2013.

Leif Kobbelt, Marc Stamminger, and Hans-Peter Seidel. Using Subdivision on Hierarchical Data to Reconstruct Radiosity Distribution. *Computer Graphics Forum*, 1997. ISSN 1467-8659.

Bongjin Koo, Wilmot Li, JiaXian Yao, Maneesh Agrawala, and Niloy J. Mitra. Creating works-like prototypes of mechanical objects. *ACM Trans. Graph.*, 33(6):217:1–217:9, November 2014. ISSN 0730-0301. doi: 10.1145/2661229.2661289. URL `http://doi.acm.org/10.1145/2661229.2661289`.

Yuki Koyama, Shinjiro Sueda, Emma Steinhardt, Takeo Igarashi, Ariel Shamir, and Wojciech Matusik. Autoconnect: Computational design of 3d-printable connectors. *ACM Trans. Graph.*, 34(6):231:1–231:11, October 2015. ISSN 0730-0301.

Vladislav Kraevoy and Alla Sheffer. Cross-parameterization and compatible remeshing of 3d models. In *Siggraph 2004*, pages 861–869. ACM, 2004.

Vladislav Kraevoy, Alla Sheffer, Ariel Shamir, and Daniel Cohen-Or. Non-homogeneous resizing of complex models. *ACM Transactions on Graphics*, 27(5):111:1–111:9, 2008.

Frank Kursawe. A variant of evolution strategies for vector optimization. In *Proceedings of the 1st Workshop on Parallel Problem Solving from Nature*, PPSN I, pages 193–197, London, UK, UK, 1991. Springer-Verlag. ISBN 3-540-54148-9. URL `http://dl.acm.org/citation.cfm?id=645821.670214`.

Timothy R Langlois, Steven S An, Kelvin K Jin, and Doug L James. Eigenmode compression for modal sound models. *ACM Trans. Graph.*, 33(4):40, 2014.

Manfred Lau, Akira Ohgawara, Jun Mitani, and Takeo Igarashi. Converting 3d furniture models to fabricatable parts and connectors. *ACM Transactions on Graphics*, 30(4):85, 2011.

Aaron WF Lee, David Dobkin, Wim Sweldens, and Peter Schröder. Multiresolution mesh morphing. In *Siggraph 1999*, pages 343–350. ACM, 1999.

Seungyong Lee, George Wolberg, and Sung Yong Shin. Scattered data interpolation with multilevel b-splines. *IEEE transactions on visualization and computer graphics*, 3(3):228–244, 1997.

Jeffrey I Lipton, Zachary Manchester, and Daniela Rus. Planning cuts for mobile robots with bladed tools. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 572–579. IEEE, 2017.

Jeffrey I Lipton, Adriana Schulz, Andrew Spielberg, Luis Trueba, Wojciech Matusik, and Daniela Rus. Robot assisted carpentry for mass customization. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, 2018.

Frank Losasso, Frédéric Gibou, and Ron Fedkiw. Simulating water and smoke with an octree data structure. *ACM Trans. Graph.*, 23(3):457–462, August 2004. ISSN 0730-0301.

James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. A compiler for 3d machine knitting. *ACM Trans. Graph.*, 35(4):49:1–49:11, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925940. URL `http://doi.acm.org/10.1145/2897824.2925940`.

Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3d-printable robotic creatures. *ACM Trans. Graph.*, 34(6), October 2015a. ISSN 0730-0301.

Vittorio Megaro, Bernhard Thomaszewski, Maurizio Nitti, Otmar Hilliges, Markus Gross, and Stelian Coros. Interactive design of 3D-printable robotic creatures. *ACM Transactions on Graphics (TOG)*, 34(6):216, 2015b.

Ankur Mehta, Joseph DelPreto, and Daniela Rus. Integrated codesign of printable robots. *Journal of Mechanisms and Robotics*, 7:021015, 2015.

A. Messac, A. Ismail-Yahaya, and C.A. Mattson. The normalized normal constraint method for generating the pareto frontier. *Structural and Multidisciplinary Optimization*, 25(2): 86–98, Jul 2003. ISSN 1615-1488. doi: 10.1007/s00158-002-0276-1. URL `https://doi.org/10.1007/s00158-002-0276-1`.

Niloy J Mitra, Natasha Gelfand, Helmut Pottmann, and Leonidas Guibas. Registration of point cloud data from a geometric optimization perspective. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH symposium on Geometry processing*, pages 22–31. ACM, 2004.

Yuki Mori and Takeo Igarashi. Plushie: An interactive design system for plush toys. *ACM Transactions on Graphics*, 26(3):45:1–45:8, 2007.

Przemyslaw Musialski, Thomas Auzinger, Michael Birsak, Michael Wimmer, and Leif Kobbelt. Reduced-order shape optimization using offset surfaces. *ACM Trans. Graph.*, 34 (4), July 2015. ISSN 0730-0301.

Przemyslaw Musialski, Christian Hafner, Florian Rist, Michael Birsak, Michael Wimmer, and Leif Kobbelt. Non-linear shape optimization using local subspace projections. *ACM Trans. Graph.*, 35(4), July 2016. ISSN 0730-0301.

Liangliang Nan, Ke Xie, and Andrei Sharf. A search-classify approach for cluttered indoor scene understanding. *ACM Trans. Graph.*, 31(6):137:1–137:10, November 2012. ISSN 0730-0301.

Robert Osada, Thomas Funkhouser, Bernard Chazelle, and David Dobkin. Matching 3d models with shape distributions. In *Proceedings of the International Conference on Shape Modeling & Applications*, SMI '01, pages 154–, Washington, DC, USA, 2001. IEEE Computer Society. ISBN 0-7695-0853-7. URL `http://dl.acm.org/citation.cfm?id=882486.884103`.

Maks Ovsjanikov, Wilmot Li, Leonidas J. Guibas, and Niloy J. Mitra. Exploration of continuous variability in collections of 3d shapes. *ACM Transactions on Graphics*, 30(4):33, 2011.

Jay Patel and Matthew I Campbell. An approach to automate and optimize concept generation of sheet metal parts by topological and parametric decoupling. *Journal of Mechanical Design*, 132(5):051001, 2010.

Helmut Pottmann and Michael Hofer. *Geometry of the squared distance function to curves and surfaces*. Springer, 2003.

Helmut Pottmann, Stefan Leopoldseder, and Michael Hofer. Registration without icp. *Computer Vision and Image Understanding*, 95(1):54–71, 2004.

Emil Praun, Wim Sweldens, and Peter Schröder. Consistent mesh parameterizations. In *Siggraph 2001*, pages 179–184. ACM, 2001.

Romain Prévost, Emily Whiting, Sylvain Lefebvre, and Olga Sorkine-Hornung. Make it stand: Balancing shapes for 3d fabrication. *ACM Trans. Graph.*, 32(4):81:1–81:10, July 2013. ISSN 0730-0301.

Sam T. Roweis and Lawrence K. Saul. Nonlinear dimensionality reduction by locally linear embedding. *Science*, 290(5500):2323–2326, December 2000. doi: 10.1126/science.290.5500.2323.

Greg Saul, Manfred Lau, Jun Mitani, and Takeo Igarashi. Sketchchair: an all-in-one chair design system for end users. In *Proceedings of the fifth international conference on tangible, embedded, and embodied interaction*, TEI '11, pages 73–80, 2011.

John Schreiner, Arul Asirvatham, Emil Praun, and Hugues Hoppe. Inter-surface mapping. In *ACM Trans. Graph.*, volume 23. ACM, 2004.

Adriana Schulz, Ariel Shamir, David I. W. Levin, Pitchaya Sitthi-amorn, and Wojciech Matusik. Design and fabrication by example. *ACM Transactions on Graphics*, 33(4), 2014.

Adriana Schulz, Ariel Shamir, Ilya Baran, David I. W. Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. Retrieval on parametric shape collections. *ACM Transactions on Graphics*, 36(1), 2017a.

Adriana Schulz, Cynthia Sung, Andrew Spielberg, Wei Zhao, Yu Cheng, Eitan Grinspun, Daniela Rus, and Wojciech Matusik. Interactive robogami : An end-to-end system for design of robots with ground locomotion. *The International Journal of Robotics Research (IJRR)*, 2017b.

Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinpun, and Wojciech Matusik. Interactive design space exploration and optimization for cad models. *ACM Transactions on Graphics*, 36(4), 2017c.

Adriana Schulz, Harrison Wang, Eitan Grinpun, Justin Solomon, and Wojciech Matusik. Interactive exploration of design trade-offs. *ACM Transactions on Graphics*, 37(4), 2018.

Thomas W. Sederberg, Jianmin Zheng, Almaz Bakenov, and Ahmad Nasri. T-splines and t-nurccs. *ACM Trans. Graph.*, 22(3), July 2003. ISSN 0730-0301.

Rajsekhar Setaluri, Mridul Aanjaneya, Sean Bauer, and Eftychios Sifakis. Spgrid: A sparse paged grid structure applied to adaptive smoke simulation. *ACM Trans. Graph.*, 33(6), November 2014. ISSN 0730-0301.

Chao-Hui Shen, Hongbo Fu, Kang Chen, and Shi-Min Hu. Structure recovery by part assembly. *ACM Trans. Graph.*, 31(6):180:1–180:11, November 2012a. ISSN 0730-0301.

Chao-Hui Shen, Hongbo Fu, Kang Chen, and Shi-Min Hu. Structure recovery by part assembly. *ACM Transactions on Graphics*, 31(6), 2012b.

Philip Shilane, Patrick Min, Michael Kazhdan, and Thomas Funkhouser. The princeton shape benchmark. In *Proceedings of the Shape Modeling International 2004*, pages 167–178, 2004.

SHREC. 3d shape retrieval contest at eurographics. `http://3dor2014.ensea.fr/SHREC2014.html`, 2014. Accessed: 2015-06-02.

Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. Geosemantic snapping for sketch-based modeling. *Computer Graphics Forum*, 32(2):245–253, 2013. ISSN 1467-8659. doi: 10.1111/cgf.12044. URL `http://dx.doi.org/10.1111/cgf.12044`. Proceedings of Eurographics 2013.

Maria Shugrina, Ariel Shamir, and Wojciech Matusik. Fab forms: Customizable objects for fabrication with validity and geometry caching. *ACM Trans. Graph.*, 34(4):100:1–100:12, July 2015. ISSN 0730-0301.

Mélina Skouras, Bernhard Thomaszewski, Peter Kaufmann, Akash Garg, Bernd Bickel, Eitan Grinspun, and Markus Gross. Designing inflatable structures. *ACM Trans. Graph.*, 33 (4):63:1–63:10, July 2014. ISSN 0730-0301. doi: 10.1145/2601097.2601166. URL `http://doi.acm.org/10.1145/2601097.2601166`.

Mélina Skouras, Stelian Coros, Eitan Grinspun, and Bernhard Thomaszewski. Interactive surface design with interlocking elements. *ACM Trans. Graph.*, 34(6), October 2015. ISSN 0730-0301.

Seungmoon Song, Joohyung Kim, and Katsu Yamane. Development of a bipedal robot that walks like an animation character. In *2015 IEEE International Conference on Robotics and Automation*, pages 3596–3602. IEEE, 2015.

Anuj Srivastava, Shantanu H Joshi, Washington Mio, and Xiuwen Liu. Statistical shape analysis: Clustering, learning, and testing. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 27(4):590–602, 2005.

Ian Stroud. *Boundary representation modelling techniques.* Springer Science & Business Media, 2006.

N Sukumar and EA Malsch. Recent advances in the construction of polygonal finite element interpolants. *Archives of Computational Methods in Engineering*, 13(1):129–163, 2006.

Timothy Sun and Changxi Zheng. Computational design of twisty joints and puzzles. *ACM Trans. Graph.*, 34(4):101:1–101:11, July 2015. ISSN 0730-0301. doi: 10.1145/2766961. URL `http://doi.acm.org/10.1145/2766961`.

Jerry O. Talton, Yu Lou, Steve Lesser, Jared Duke, Radomír Měch, and Vladlen Koltun. Metropolis procedural modeling. *ACM Trans. Graph.*, 30(2):11:1–11:14, April 2011. ISSN 0730-0301. doi: 10.1145/1944846.1944851. URL `http://doi.acm.org/10.1145/1944846.1944851`.

Johan W. Tangelder and Remco C. Veltkamp. A survey of content based 3d shape retrieval methods. *Multimedia Tools Appl.*, 39(3):441–471, 2008.

Joshua B. Tenenbaum, Vin de Silva, and John C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319, 2000.

Bernhard Thomaszewski, Stelian Coros, Damien Gauge, Vittorio Megaro, Eitan Grinspun, and Markus Gross. Computational design of linkage-based characters. *ACM Transactions on Graphics*, 33(4):64, 2014.

Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive couture for interactive garment modeling and editing. *ACM Trans. Graph.*, 30(4):90:1–90:12, July 2011. ISSN 0730-0301.

Nobuyuki Umetani, Takeo Igarashi, and Niloy J. Mitra. Guided exploration of physically valid shapes for furniture design. *ACM Trans. Graph.*, 31(4), 2012.

Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. Pteromys: Interactive design and optimization of free-formed free-flight model airplanes. *ACM Trans. Graph.*, 33(4):65:1–65:10, July 2014. ISSN 0730-0301.

Oliver Van Kaick, Hao Zhang, Ghassan Hamarneh, and Daniel Cohen-Or. A survey on shape correspondence. In *Computer Graphics Forum*, volume 30, pages 1681–1707. Wiley Online Library, 2011.

Nuno Vasconcelos and Andrew Lippman. A multiresolution manifold distance for invariant image similarity. *Multimedia, IEEE Transactions on*, 7(1):127–142, 2005.

Elif Vural and Pascal Frossard. Discretization of parametrizable signal manifolds. *Image Processing, IEEE Transactions on*, 20(12):3621–3633, 2011.

Cheng-Hua Wang and Robert H Sturges. Bendcad: a design system for concurrent multiple representations of parts. *Journal of Intelligent Manufacturing*, 7(2):133–144, 1996.

Wenping Wang, Helmut Pottmann, and Yang Liu. Fitting b-spline curves to point clouds by curvature-based squared distance minimization. *ACM Transactions on Graphics (ToG)*, 25(2):214–238, 2006.

Emily Whiting, Hijung Shin, Robert Wang, John Ochsendorf, and Frédo Durand. Structural optimization of 3d masonry buildings. *ACM Trans. Graph.*, 31(6):159:1–159:11, 2012.

Kai Xu, Hanlin Zheng, Hao Zhang, Daniel Cohen-Or, Ligang Liu, and Yueshan Xiong. Photo-inspired model-driven 3d object modeling. *ACM Transactions on Graphics*, 30(4):80, 2011.

Kun Xu, Yong Li, Tao Ju, Shi-Min Hu, and Tian-Qiang Liu. Efficient affinity-based edit propagation using k-d tree. In *Siggraph Asia 2009*, pages 118:1–118:6. ACM, 2009. ISBN 978-1-60558-858-2.

Yong-Liang Yang, Yi-Jun Yang, Helmut Pottmann, and Niloy J. Mitra. Shape space exploration of constrained meshes. *ACM Trans. Graph.*, 30(6):124:1–124:12, December 2011. ISSN 0730-0301. doi: 10.1145/2070781.2024158. URL `http://doi.acm.org/10.1145/2070781.2024158`.

Hironori Yoshida, Takeo Igarashi, Yusuke Obuchi, Yosuke Takami, Jun Sato, Mika Araki, Masaaki Miki, Kosuke Nagata, Kazuhide Sakai, and Syunsuke Igarashi. Architecture-scale human-assisted additive manufacturing. *ACM Trans. Graph.*, 34(4):88:1–88:8, July 2015. ISSN 0730-0301. doi: 10.1145/2766951. URL `http://doi.acm.org/10.1145/2766951`.

M. Zeleny. Compromise programming. In J. Cochrane and M. Zeleny, editors, *Multiple Criteria Decision Making*, pages 262–301. University of South Carolina Press, Columbia, 1973.

J. Zhang and L. Xing. A survey of multiobjective evolutionary algorithms. In *2017 IEEE International Conference on Computational Science and Engineering (CSE) and IEEE International Conference on Embedded and Ubiquitous Computing (EUC)*, volume 1, pages 93–100, July 2017. doi: 10.1109/CSE-EUC.2017.27.

Qingfu Zhang and Hui Li. Moea/d: A multiobjective evolutionary algorithm based on decomposition. *IEEE Transactions on evolutionary computation*, 11(6):712–731, 2007.

Youyi Zheng, Hongbo Fu, Daniel Cohen-Or, Oscar Kin-Chung Au, and Chiew-Lan Tai. Component-wise controllers for structure-preserving shape manipulation. *Computer Graphics Forum*, 30(2):563–572, 2011a.

Youyi Zheng, Hongbo Fu, Daniel Cohen-Or, Oscar Kin-Chung Au, and Chiew-Lan Tai. Component-wise controllers for structure-preserving shape manipulation. In *Computer Graphics Forum*, volume 30, pages 563–572. Wiley Online Library, 2011b.

Aimin Zhou, Bo-Yang Qu, Hui Li, Shi-Zheng Zhao, Ponnuthurai Nagaratnam Suganthan, and Qingfu Zhang. Multiobjective evolutionary algorithms: A survey of the state of the art. *Swarm and Evolutionary Computation*, 1(1):32–49, 2011.

Bo Zhu, Mélina Skouras, Desai Chen, and Wojciech Matusik. Two-scale topology optimization with microstructures. *ACM Trans. Graph.*, 36(5):164:1–164:16, July 2017. ISSN 0730-0301. doi: 10.1145/3095815. URL `http://doi.acm.org/10.1145/3095815`.

Lifeng Zhu, Weiwei Xu, John Snyder, Yang Liu, Guoping Wang, and Baining Guo. Motion-guided mechanical toy modeling. *ACM Trans. Graph.*, 31(6):127:1–127:10, November 2012. ISSN 0730-0301. doi: 10.1145/2366145.2366146. URL `http://doi.acm.org/10.1145/2366145.2366146`.

E. Zitzler, K. Deb, and L. Thiele. Comparison of Multiobjective Evolutionary Algorithms: Empirical Results. *Evolutionary Computation*, 8(2):173–195, 2000.

Denis Zorin and Peter Schroder. Subdivision for modeling and animation. In *Siggraph 2000 Courses*. ACM, 2000.