Design for Sustainable Fabrication

Yuxuan Mei

A dissertation
submitted in partial fulfillment of the
requirements for the degree of

Doctor of Philosophy

University of Washington
2025

Reading Committee:
Adriana Schulz, Chair
Jennifer Mankoff
Vikram Iyer
Afroditi Psarra

Program Authorized to Offer Degree:
Computer Science and Engineering

University of Washington

Abstract

Design for Sustainable Fabrication

Yuxuan Mei

Chair of the Supervisory Committee:

Adriana Schulz
Computer Science and Engineering

Design tools play a crucial role in how effectively the advances in manufacturing can be utilized to create products of unprecedented complexity. The usability of these tools directly impacts who can design, customize, and ultimately enhance the performance of products. Meanwhile, the revolution in manufacturing is fundamentally changing what can be made and by whom, enabling on-demand production of complex and functional objects. Despite its ever-increasing potential, manufacturing continues to consume considerable natural resources [43] and produce significant amounts of waste [42] and carbon emissions [59]. This raises a critical research question: how can we foster sustainable fabrication practices in this rapidly evolving landscape? In this dissertation, we address the challenge by developing computational design systems that make it easier for users to create sustainable designs, thereby reducing material usage and waste at the source. Our approach centers on creating domain-specific abstractions for design systems that enable more effective exploration of the design space considering sustainability and expression of users' design intent. We show how this approach can be applied to various fabrication domains, including electronics, home accessories, and textiles, and discuss future work directions and potential impact beyond fabrication.

Acknowledgements

Not all journeys lead to the stars. But looking back, my PhD journey was full of serendipitous encounters.

I want to begin by thanking my advisor, Adriana Schulz, whom I look up to as an exemplary researcher, a role model, and who continues to inspire me with her work ethic and passion. I am fortunate to have learned so much from supportive collaborators and mentors. I thank Jennifer Mankoff, Vikram Iyer, and Afroditi Psarra for serving on my committee, all the machine knitting friends (Jen, Megan, Jim, Vidya, Jenny, Tianhong), MechE friends (Jeff, Daniel, and others in Jeff's group), and electronics experts (Vikram, Zhihan, Zachary) for showing me the way in your domain expertise, and Etienne Vouga for tirelessly teaching me simulation and how to debug cmake. I am grateful for the CS Education community at UW and all the mentorship and support I have received. And I would not have made it without everyone in CDG (a.k.a. Adriana's group, thanks Ben, Dalton, Haisen, Amy, Dan, Milin, James, Felix, Jack, and Nat!), RCR, GRAIL, WiGRAPH, the CSE staff and advising team, and my fellow PhD friends (Alice, Althea, Amy, Audrey, Jingwei, Judy, Liwei, Ruotong, Sitong, and Wei).

Although I was only home for one month total during the six years, words could not express how much my parents have supported me. Being away from home is a challenge and I am grateful that video conferencing and the Internet connect us across the globe. I only feel fortunate that I got to explore so many things, thanks to my parents, whose love and support gave me courage to go on and face the difficulties. Thank you, mom and dad, for being my best friends and teachers in life.

DEDICATION

To Guiping Ni and Xin Mei

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Our Impact on the Environment

The industrial revolutions fundamentally transformed the world economy and how people live and work. Machines greatly improved the efficiency of manufacturing goods, which in turn significantly bettered our living conditions. In the past few decades, new advancements in manufacturing like 3D printing in aerospace and automotive industries, whole-garment knitting machines in apparel, and flexible substrates in electronics, is further changing what can be made and by whom, enabling on-demand production of complex and functional objects.

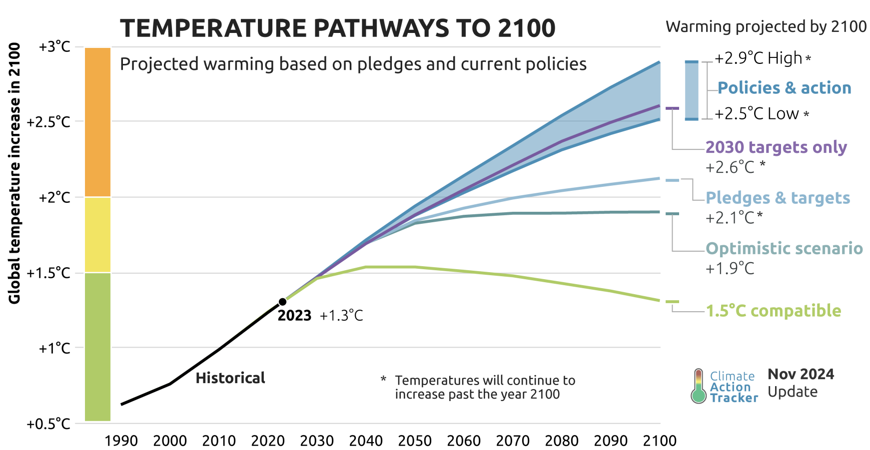


Figure 1.1: Global temperature increase by 2100 according to Climate Action Tracker's warming projections [185].

Despite its ever-increasing potential, manufacturing continues to consume considerable natural resources [43] and produce significant amounts of waste [42] and greenhouse gas emissions [59]. As we continue to witness evidence of deteriorating ecosystems due to human activities including manufacturing, sustainability is on more and more people's minds. For example, to curb global warming, global leaders have come together to set the goal of limiting the temperature increase to 1.5°C with the Paris Agreement [188] in 2015. However, the real-world situations after 10 years of the agreement show that the climate is warming much faster than the governments are acting [186] (Figure 1.1). In order to protect the environment we depend on, it is critical that we take more immediate actions to make manufacturing more sustainable.

From the perspective of computational design and fabrication, design tools play a crucial role in how effectively manufacturing can be utilized to create products of unprecedented complexity and functionality. The usability of these tools directly impacts who can design, customize, and ultimately enhance the performance of products, including how sustainable they will be, because the design of a product often directly impacts how it is manufactured and what materials will be used. Through developing innovative computational design algorithms and user interfaces, we hope to broaden the range of who can participate and thus the resulting panoply of designs, and in the meantime, foster sustainability awareness, enabling and encouraging users to create more environmentally friendly designs. Therefore, a critical research question emerges: how can we build computational design systems that foster sustainable manufacturing practices? This dissertation aims to address the question from a computational design perspective and formalizes approaches for building design systems that empower sustainable manufacturing.

1.2 Design Systems that Support Sustainability

We first formalize what traits a design system need to have to support sustainability. Take the scenario illustrated in Figure 1.2 as an example. Suppose a designer is trying to create a design of an electronic device. The various design choices, including the ICs (integrated circuits) and components to use, the PCB (printed circuit board) design, the logic in the circuit, and also the physical form factor of the device, form a design space. Then creating a design is as if the designer is picking a point in this design space. Usually, designers pick the point based on some performance metrics that evaluate how the design would perform after it is deployed in the real world, which might include the manufacturing costs and the functional efficiency.

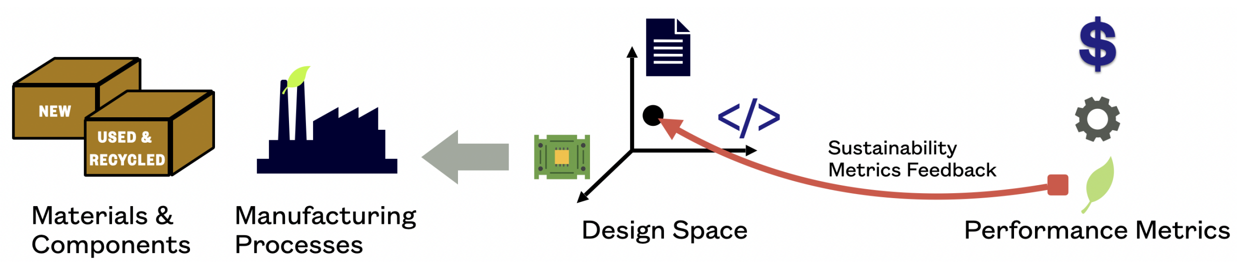


Figure 1.2: Diagram that shows how can design systems support more sustainable manufacturing practices. The example scenario here is a design space of electronic products that involves designs axes such as ICs, subcomponents, and the circuit logic.

To support designers to create a more sustainable design, it is therefore important for design systems to help designers understand how sustainable a design is. By making sustainability metrics available to designers, we can enable designers to make more sustainable design choices guided by these metrics and stay more informed throughout the design process.

In addition to incorporating sustainability feedback into the design loop, we can also expand design systems' capabilities to consider more sustainable design constraints. There are different kinds of sustainable design constraints. For example, one category is on the kinds of materials being used. The design space might be constrained when we consider only materials that are more sustainable themselves (e.g., biodegradable, non-toxic), or when we consider using more recycled materials rather than only new materials. In addition to the materials used, which directly impact the design space, a hidden factor that poses constraints on the design space is the manufacturing processes. Different manufacturing processes place different constraints on a design space, and in fact, some fabrication processes are inherently more energy efficient or material efficient than others. For example, in garment production, the cut-and-sew method generates more waste than the fully-fashioned knitting method because cut-and-sew cuts garment panels from a larger piece of fabric, leaving fabric leftovers, while fully-fashioned knitting can create the exact panels without waste. Thus, design systems ought to take these constraints into account so that designers could create designs that can be fabricated in a more sustainable way.

To summarize, design systems that foster sustainable manufacturing practices should (1) allow designers to consider feedback on sustainability metrics, and (2) enable designers to create designs with sustainability constraints.

1.3 Design Abstractions for Sustainable Manufacturing Practices

An important backbone for a design system is the design representation used when devising the design system. Design representation encapsulates domain-specific knowledge and distills prominent design patterns. Usually, a design representation also abstracts away low-level details and therefore presents a more focused, higher-level view for decision making. A good design representation makes it easier to embed design desiderata, enabling more effective exploration and search of the design space.

Thus, the key to create design systems for sustainable manufacturing is to find the proper design abstractions that (1) allow easy evaluation of environmental impact metrics and (2) encapsulate the constraints posed by sustainable manufacturing practices. Embedding domain knowledge about how to evaluate sustainability metrics of a design is crucial for computing complex environmental impact metrics and enables optimization methods for searching the design space. Moreover, a good representation of the design space also enables easier expression of users' design intent. With abstractions that capture meaningful aspects of the design, users can interact with higher-level constructs rather than specifying every detail in the design process. This could in turn help users navigate the landscape of design tradeoffs between sustainability metrics and other design goals such as costs and performance.

Following this approach, we build design tools and systems to support creating sustainable designs, combining insights from programmable abstractions with geometric modeling. We show how material usage and environmental impacts can be taken into account during the design process with the suitable representations, and how to create design tools that make it easier for users to create sustainable designs in different fabrication domains.

1.4 Roadmap

This dissertation is organized as follows:

We start with reviewing relevant background on sustainable design in Chapter 2, focusing on prior computational design research with a focus on sustainability.

Next, Chapter 3 focuses on design systems that allow designers to consider sustainability metrics, and Chapter 4 focuses on design systems that take into account different types of sustainability constraints.

The previous chapters build up to Chapter 5, which introduces a design system that both incorporates sustainability feedback and considers sustainable manufacturing constraints.

Lastly, we conclude with a discussion on the limitations and future work directions in Chapter 6.

1.5 Contributions

As discussed earlier, the key insight underlying the work in this dissertation is the use of domain-specific design representations to enable efficient evaluation and optimization of designs and exploration of a design space. It is a common strategy to leverage domain knowledge to construct simplifying models when solving complex problems. As such, we place an emphasis on how we developed design abstractions tailored to specific fabrication domains.

The core contributions of this dissertation are representations, algorithms, and design systems that serve as case studies in applying this problem-solving strategy to support sustainable manufacturing:

1. An abstraction of PCB designs and a comparison tool for comparative assessment of environmental impacts for PCBs (Chapter 3, contributed to the comparison algorithm, comparison tool implementation, and case study evaluations in Zhang et al. [218])
2. A representation of knit objects that enables knitters without machine knitting expertise to create designs that can be fabricated with machine knitting, which is a material-efficient fabrication process (Chapter 4 Section 4.1, contributed to the knitter-friendly quad meshing method and physical evaluations in Jones et al. [97])
3. A representation and design tool for designing hacks with everyday objects found at home (Chapter 4 Section 4.2, Mei et al. [128]).
4. A representation and design tool for designing visually pleasing while material efficient patchwork from scrap fabrics (Chapter 5, Mei et al. [129]).

Chapter 2

Background and Related Work

2.1 The Manufacturing Industry

In 2015, Global leaders have come together to curb global warming, setting the goal of limiting the temperature increase to 1.5°C with the Paris Agreement [188]. To realize this goal, it is necessary to reduce the greenhouse gas (GHG) emissions. Among the main contributors to GHG emissions, *the industry*, that produces the materials and goods we consume daily, comes second at around 25.4% worldwide just after the power industry (i.e., electricity and heat) [61]. According to the US Environmental Protection Agency, the industry sector is responsible for 30% of total carbon emissions in the US after distributing the power sector's emissions to specific end-use sectors, paralleling the 30% of the commercial/residential sector, and 29% of transport [60].

Several opportunities exist for reducing emissions for the industry sector as summarized by the US EPA, including improving energy efficiency in manufacturing, switching to cleaner energy sources, recycling, and increasing awareness and action through training for companies and workers [59]. Most existing efforts focus on decarbonization, particularly through improving energy efficiency or transitioning to cleaner energy sources. In contrast, relatively fewer efforts target the materials aspect of sustainability, among which recyling is one of the well-recognized strategies.

2.2 Theories on Design and Sustainability

Prior work has been done on the theoretical side of design and sustainability and we briefly discuss two theories in this section.

2.2.1 Design for Sustainability

The field of DfS emerged in the 1980s when the seminal work "Design for the Real World" [146] introducing considerations of resources and energy usage into the realm of design was published. Ceschin and Gaziulusoy [30] provided a chronological overview of DfS and they were the first to develop a framework that describes how the field has evolved in the past years. Under this framework, the DfS efforts have shifted from a product focus towards a socio-technical focus. In other words, it is moving from the product-level technological innovation that is considered more insular and technology-focused, to more people-centered and systemic approaches to drive the change for more sustainable practices.

Indeed, we need more than motivations for end users to adopt sustainable behavior; the effort towards sustainability has to also involve multiple stakeholders. However, innovating at a product/technology level is still valuable because this contributes directly to reducing environmental impact, and is getting integrated into more systemic approaches to amplify the impact. Therefore, from this perspective, it is crucial not only to provide means for users to have a better understanding of the environmental impact of their decisions and behaviors, but also to promote broader adoption of sustainable products and technologies by engaging a wider range of stakeholders.

2.2.2 Sustainable Interaction Design

In 2007, Blevis [12] introduced the theory of SID. SID places a focus on the interaction design instead of the object being designed. More specifically, it considers "design as an act of choosing among or informing choices of future ways of being" and envisions that SID can guide people towards a viable future of sustainable living. The work outlines a set of design values, methods, and reasoning from the perspective of sustainability and discusses open research questions.

The two principles that were discuss are *linking invention & disposal* and *promoting renewal & reuse*, which are both directly related to the overarching goal of this dissertation (which is to create design systems that support sustainable manufacturing). Throughout the dissertation, we will see how similar ideas form and how we develop them for specific domains.

2.3 Sustainable Manufacturing from a Design Perspective

One of the key principles of sustainable design [55] is to *get more product or service out of less material*. This can mean reusing existing materials and products or reducing the materials required to manufacture new products. Less material used often means less environmental impact. From the environmental impact perspective, we can also interpret this as "getting more product or service out of less environmental impact" in a broader sense. In this dissertation, the works discussed implicitly follow this principle and roughly correspond to these interpretations. We now survey prior computational design research and manufacturing innovations relevant to this sustainable design principle.

2.3.1 Using more sustainable materials and components

Here, "sustainable" means less environmental impact. For example, materials and components that can be manufactured with less energy, that have less embodied carbon, or that can be more easily recycled and reused. Many product-level and technology-level DfS innovations focus on developing more sustainable materials that can be more easily recycled or have less environmental impact. For example, biodegradable materials [191], like vitrimers [217] and fiber-based composites [6], have been used in electronics.

Computational design tools can also guide users in making more informed decisions in the design process by providing information on the environmental impact of the design or components in the design. For example, Echenagucia et al. [132, 133] have developed methods for exploring trade-offs between embodied carbon and operational carbon of building designs.

2.3.2 Recycling and reusing existing objects

Recycling and reuse reduce some need to manufacture new materials and products by repurposing existing materials and products for a second use. Recycling is one of the most important ways to achieve sustainable manufacturing via a “circular economy”. Being able to recycle parts or all of an item for some other use is essentially getting more service out of that item. In addition to recycling, materials can be upcycled or downcycled. Take fabrics as an example. Upcycling means that the fabric is still in good condition and can be used in smaller-scale projects like pillow covers [178] or fashion designs [44], while downcycling could mean that the fabric is losing its structural integrity and will be broken down into insulation materials [103, 63].

Recently, recycling and sustainability have been a growing focus of the fabrication research community [207, 208]. Prior work explored computational tools to guide users in reusing plastic bags [35], yarns [205], or wasted materials from manufacturing processes [50, 125, 126]. For electronics, various tools focusing on reusing e-waste [104, 122, 121, 209, 210] and the unmaking of electronics [123] have been explored. In 3D printing, researchers developed methods to reduce the amount of support materials needed by reusing existing objects [197] or reusable structures [206], and 3D printing itself has been used to fix broken objects [183, 107] to reduce waste. Another line of work is to augment existing objects with fabrication to achieve repurposing [45, 153, 73], such as by generating structures for re-interfacing with robot arms, legacy physical interfaces, or appliances. Chen et al. [33, 34, 32] use 3D printing to augment existing objects with additional functionality (some involving mechanisms), while Arabi et al. [4, 5] and Li et al. [115, 114, 116] focus more on augmenting robots using everyday objects or mechanisms to help robots manipulate objects.

2.3.3 Using less material

This is a direct interpretation of the sustainable design principle. One direction is to optimize designs to require less material to fabricate but still achieve comparable performance (e.g., through topology optimization). For example, in the garment industry and among hobbyists, zero-waste pattern-making attracts a lot of interest [157, 26, 79, 216]. Other applications include carpentry, where Wu et al. [202] propose a design system for carpentered objects based on a novel domain-specific language that can help optimize material costs, and 3D printing, where prior work studied methods for optimizing material usage for support structures [190, 159, 52]. Another direction is to use fabrication techniques that inherently waste less material. For example, knitted garments are produced with minimal wasted yarn [180], but compared to knitting, sewn garments inherently require cutting and sewing, which creates fabric leftovers. From a computational design perspective, we can guide users in using such fabrication processes by providing users with easier-to-use design tools.

Chapter 3

Design with Sustainability Feedback

In the manufacturing industry, standards and methodologies have been developed to measure the environmental impact (EI) of products. On one hand, EI analyses help companies to meet regulations on environmental impact reporting, and on the other hand, insights can actually surface from these analyses on how to create more sustainable designs. If product designers need to consider sustainability while designing, they need to understand the EI of the designs they are creating, ideally during the design process, so that they can make more informed design decisions like choosing more sustainable materials and components.

In this chapter, we will take a closer look at the ICT (Information and Communication Technology) industry, given that it is both a major contributor to carbon emissions and has a relatively mature system for evaluating EIs. We begin by introducing some background on the EI evaluation method LCA (Life-Cycle Assessment), and then identify current challenges associated with the method. We conclude with a novel workflow that utilizes domain knowledge on electronics and LCAs to enable designers to compare sustainability metrics across designs. This approach works even when complete EI information about materials and components is unavailable during the design stage.

3.1 Life-Cycle Assessment

Performing a Life-Cycle Assessment (LCA) involves systematically evaluating the EIs associated with a product's life-cycle stages, from raw material extraction, manufacturing, distribution, use, repair and maintenance, to disposal or recycling. This analysis of the total EIs enables manufacturers and consumers to make more informed decisions. The process begins with defining the scope and the goal of the assessment, which includes identifying the product to be assessed and the system boundaries of the study as shown in Figure 3.1. Common LCA boundaries include "cradle-to-gate" assessments which account for raw material extraction through product manufacturing, and "cradle-to-grave" which also includes transport to the consumer, usage throughout the product's lifetime, and end-of-life disposal. The next step is an inventory analysis, where data is collected on every input (energy, water, and materials) and output (e.g., emissions and waste) associated with each life-cycle stage. The impact assessment phase interprets this data to understand the EIs, such as global warming potential, ocean acidification, or resource depletion. Finally, the results are interpreted to provide insights and recommendations for reducing the EI. The global warming potential is one of the most commonly used indicators for environmental impact. Since the global warming potential is usually measured in the unit of carbon dioxide equivalent (kg CO2-eq.), it is also known as carbon footprints (CF) or carbon emissions.

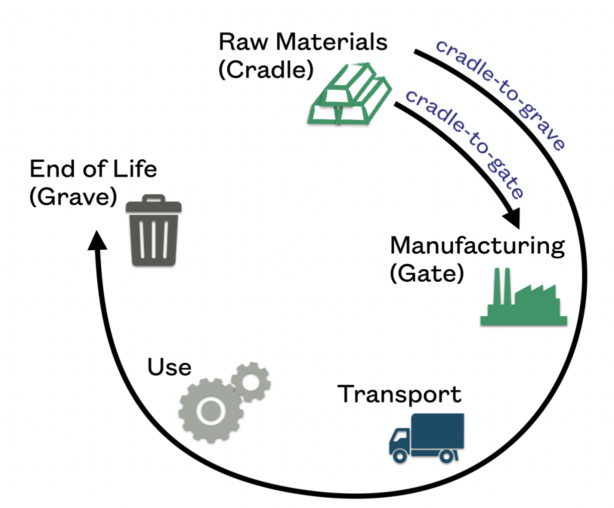


Figure 3.1: Five life-cycle stages and two common system boundaries, cradle-to-gate and cradle-to-grave.

A growing number of companies release consumer-facing product environmental impact reports, which tabulate the results of a completed LCA to report EI metrics. In the ICT industry, this process is typically completed *after* the production phase. To further complicate matters, the information reported in an LCA does not translate into clear design recommendations. Bhander et al. [11] describe this as the environmentally-conscious design process paradox, where conducting a LCA is contingent upon first completing the design, at which point it is often infeasible to make design changes based on the results of the analysis. Our semi-structured interview study [58] also validated this paradox. Indeed, there exists a timeline mismatch between the design process and the LCA process (Figure 3.2 top) in the electronics industry. But as Section 4.2 from Englhardt et al. [58] pointed out, there is also clear momentum and interest in more tightly integrating the LCA and product design timelines (Figure 3.2 bottom).

To address this challenge, the construction and architecture sectors have begun placing a consistent focus on early-stage design optimization towards more sustainable building designs [177, 132, 78, 133]. This has led to widespread integration of LCA estimation and simulation functionality directly into building information management (BIM) software, or computational tools that enable designers to explore the complex trade-offs between different building envelope designs [83]. Having access to this information during the early design stage is invaluable due to the significant cost of building construction.

Compared to the construction sector, there is currently a lack of support for early-stage EI estimation of electronic designs, but recent work has begun to explore this topic. Researchers in computer architecture

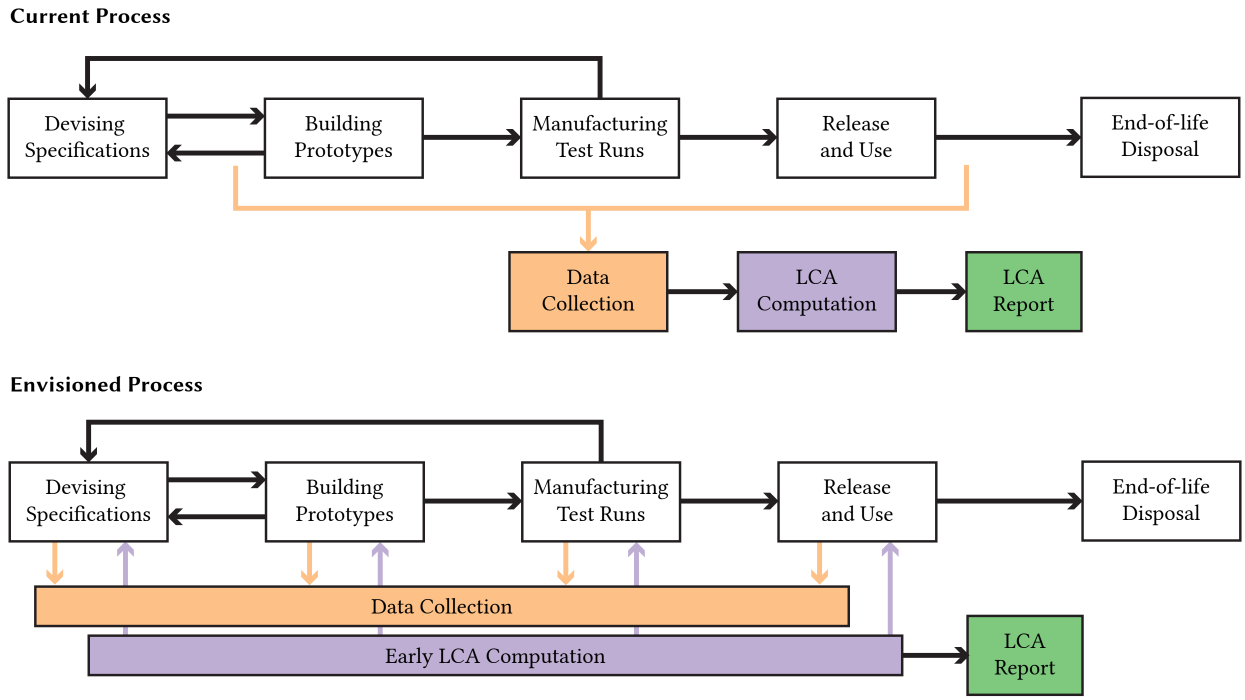


Figure 3.2: LCA integration with product development. (Top) In the current product development process, LCA data is collection begins after the final BOM is completed. This means products have already reached the market by the time the LCA report is available. (Bottom) In our envisioned product development process, LCA data collection and computation are carried out in parallel to all design stages in an integrated fashion. The preliminary LCA results can now be used to inform decision-making.

and systems have developed guidelines and carbon footprint estimation tools [74], modeling the EI of data centers and cloud computing systems [1, 147, 198, 56] and PCB designs [31]. In addition, the product carbon footprint (PCF) reports from companies usually focus on carbon footprint numbers and tend to be less thorough or do not provide actual numbers on the resources depletion or toxicity impact. To this end, researchers also start to look into the non-carbon environmental impacts of electronics and e-waste [93, 92].

3.1.1 Why is LCA hard?

At first glance, it might seem that LCA is rather straightforward because it is simply a sum of all the EI terms associated with a product, and the individual terms are computed just by multiplying a quantity value with an emissions factor value. But LCA is a far more involved process in reality, and electronic LCAs present additional challenges unique to the ICT industry.

Within the procedure of LCA, collecting the needed data to perform the necessary calculations is usually the most difficult and time consuming. This step is also called the Life-Cycle Inventory, and the resulting data is called the bill of materials (BOM). When it comes to electronics, a product could often involve hundreds

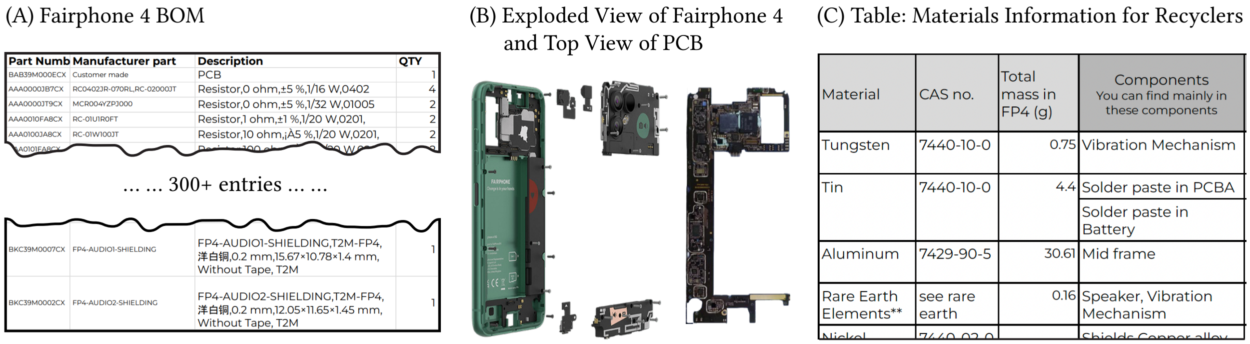


Figure 3.3: Smartphone BOM. (A) An excerpt from the manufacturing BOM for the Fairphone 4, which lists over 300 unique components. (B) An exploded view of the Fairphone 4 assembly [166]. (C) Materials information for recyclers compiled and collected through the LCA data gathering process. Both A and C are from the manufacturer-provided repair and recycling information document for the Fairphone 4 [64].

of components, and each of them requires collecting data of the raw materials used, as can be seen in the example BOM in Figure 3.3. Moreover, there are significant differences between manufacturing processes because complex chips and printed circuit boards can have quite involved fabrication steps. In addition to the challenge due to the sheer complexity, another layer of challenges comes from the communication between different stakeholders (Figure 3.4). Oftentimes, the data needed for performing an LCA is scattered across the corporate hierarchy, requiring the LCA practitioners to trace back multiple steps until they find the piece of data they need. For components sourced from external suppliers, it could be even harder to obtain information about the manufacturing processes and materials used due to information privacy concerns. For more details, please refer to Section 4.1 in Englhardt et al. [58].

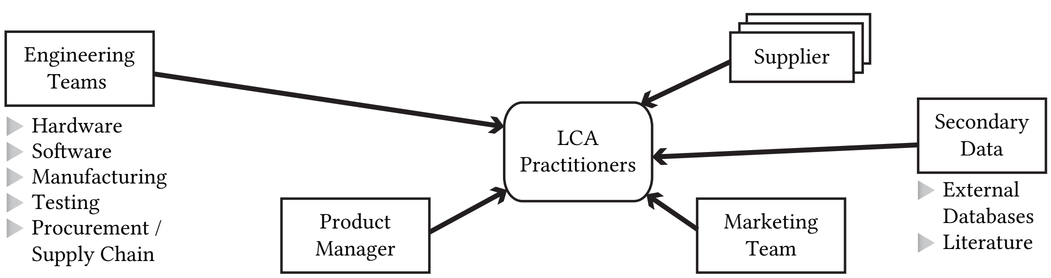


Figure 3.4: LCA data collection. LCA practitioners require input from many product stakeholders to collect the data needed to accurately model EI. Without being part of the initial design process, they often must retrace the steps it took to arrive at the final product assembly.

Related to the product's BOM is the information of emission factors, which is also required to perform an LCA. This is the key information needed for the next step in LCA, Environmental Impact Assessment. As the ICT industry advances, new chips and components are being developed and used. But this also means that LCA software tools and databases may not have the newest information about the environmental impact factors for these new materials and components. As a result, LCA practitioners typically use industry averages as a substitute, or try to choose an emission factor of a different but close enough item using their expertise.

Since it is often hard to have complete information ready for an LCA, there exists different ways to handle the missing information. One could choose to explicitly not model the part with missing information. For example, certain analysis, known as cradle-to-gate, gate-to-gate, etc., ignores certain upstream or downstream processes [72]. Others focus on a narrower range of evaluated environmental metrics [8], which streamlines the inventory phase by reducing the inventory parameter considered in the selected categories. Or, one could use surrogate values that are available to approximate the missing values. For example, some simplify the analysis by using mass as a coarse-grained indicator of raw material used [138, 100] to perform quicker assessments. To this end, various standards [38, 39] have been discussed and proposed in order to standardize how the industry should fill in the missing information. Despite the standards, LCAs can still vary significantly for the same product due to differences in system boundaries, modeling decisions and assumptions, and data collection procedures. In fact, full LCA analyses could have error margins ranging from 19% [47] to 300% [48].

Alternative LCA Methodologies Efforts to reduce the complexity and cost of traditional LCA methods have led to the development of streamlined approaches. One recent approach that is widely adopted in the industry is the Product Attribute to Impact Algorithm (PAIA) [141], which streamlines the LCA process by statistically triaging data and mapping product attributes to EI at the module level. For example, PAIA assesses the EI of a laptop by taking in a predefined set of attributes such as its screen size, memory capacity, and printed circuit board area. Recently, several industry tools [171, 124] have adopted similar models, translating detailed component information into CFs. Additionally, tools like ACT [74] have emerged to support the embodied CF modeling of computer architectures at the IC level of design. Another approach is the Economic Input-Output Life Cycle Assessment (EIO-LCA) [80], which estimates EI based on the market price of goods, but it often lacks precision for product-specific assessments due to its reliance on aggregated sector data.

Despite these advancements, existing tools still require users to manually determine a BOM or other detailed knowledge of an object's internals as input. This manual intervention hinders the adoption of LCA in real-time decision-making contexts.

Comparative Life-Cycle Assessment Prior work has explored the idea of performing comparative LCA as a decision-making tool, enabling stakeholders to evaluate and compare the EIs of various alternatives and successors [214, 120]. Comparative LCA is also crucial for making comparative claims for publicity purposes. However, existing works compare by looking at two complete LCAs and comparing the resulting outputs. This approach presents several challenges. First, the cost and time associated with full LCA studies are often prohibitive, particularly in industries like electronics characterized by rapid product development cycles. Second, the complexity of modern supply chains, especially in sectors like electronics, makes the quantification of all processes a time-consuming and resource-intensive task [20, 194, 3]. And lastly, the requirement for extensive and often confidential data makes full LCAs daunting.

3.2 DeltaLCA: Comparative Life-Cycle Assessment for Electronics Design

As discussed earlier, it is impractical to conduct multiple complete LCAs for different design variations for designers to take sustainability into account during the design process. Thus, we propose a novel approach to enable designers to pick the electronic design with a lower carbon footprint at the PCB (printed circuit board) design stage, building on two insights:

1. This design tool is centered around empowering designers with informed decision-making during both the drafting and iteration stages. This approach recognizes that conducting conventional LCAs for each design alternative is not only costly but also impractical as search spaces become larger. We observe that informed decision-making does not require absolute metrics, but can be done instead with relative comparisons between different designs;
2. In most cases, when we do not have complete EI information, we can still leverage domain-specific knowledge of electronic manufacturing to reason about relative differences between parts or designs.

We combine these insights to develop DeltaLCA, an open-source interactive design tool that addresses the dual challenges of automating life-cycle inventory generation and data availability by performing comparative analyses for electronics. Users can input two standard PCB design files from an Electronic Design Automation (EDA) software, and the tool will guide the users in determining which one has a greater carbon footprint. DeltaLCA leverages electronics-specific LCA datasets and heuristics and tries to automatically rank the two designs, prompting users to provide additional information only when necessary, through a user interface designed to highlight missing data. This approach dramatically simplifies LCA computation by canceling out common components, automatically estimating the carbon footprint of remaining components where possible, and providing a simple interface to input new information or use heuristics to compare the remaining parts automatically.

3.2.1 System Overview

This section presents an overview of DeltaLCA, a system to assess the relative EI between two input designs $\mathcal{A}$ and $\mathcal{B}$. DeltaLCA can be used at various stages in the design process where $\mathcal{A}$ and $\mathcal{B}$ could be individual components, partial designs, or complete designs. The intended use case of our system is where a designer inputs PCB design files from an EDA software to enable seamless integration with existing workflows of an electronic designer. Next, we discuss the differences between the traditional way of comparing two designs and DeltaLCA (as illustrated in Figure 3.5).

Traditional LCA workflow As described in Section 3.1, a traditional LCA is performed in two phases, the inventory phase and the assessment phase (green and orange boxes), respectively, in Figure 3.5. Both phases are entirely performed by LCA experts. In the inventory phase, the expert lists all relevant parts of the

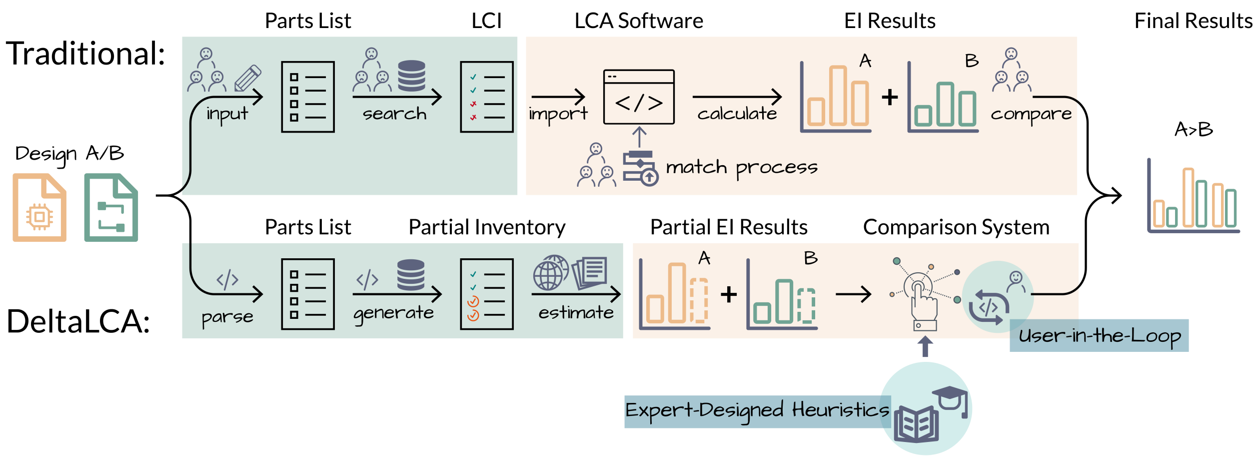


Figure 3.5: DeltaLCA enables designers to rapidly compare the environmental impact (EI) of two PCB designs. In the traditional LCA pipeline (top row), LCA experts manually generate an inventory of parts and match them to a database to calculate the EI. DeltaLCA (bottom row) first automates inventory generation and estimates partial EI for known components, then performs a user-in-the-loop *comparison* using domain-specific heuristics to determine if Design A has a greater EI than Design B.

design file to establish a Complete Parts List. Then, for each part in that list, the expert needs to fill in EI data from an LCA database, such as ecoinvent [201]. Parts, especially recent electronic parts, are often not readily available in these databases, and the expert will need to search for the best approximation using their domain knowledge, making it impossible to automate this process. In the assessment phase, the expert imports additional processes and establishes a flow diagram in an LCA software, such as GaBi/Sphera [175]. Creating a flow diagram requires both an understanding of the LCA software and knowledge about the secondary processes and resources needed to manufacture parts. This entire process is done for both input designs, resulting in two EI results. Finally, once the expert has these numbers, they can compare the two designs.

DeltaLCA workflow In DeltaLCA, we completely automate the inventory phase, where we leverage our domain knowledge to build an LCA-targeted parser of design files and automatically establish a Complete Parts List. For each part in a design file, the parser parses all the information available in public datasets and auxiliary sources, and the result of this step is a Partial LCI (see Section 3.2.2). A Partial LCI contains three kinds of parts:

1. Parts which are identical and which are present in both $\mathcal{A}$ and $\mathcal{B}$. We can cancel these parts out before the comparison algorithm.
2. Unique parts for which we have sufficient LCA information to compute their EIs.
3. Unique parts for which we have only partial information.

We then replace the two assessment phases with a single *comparative LCA phase* using the two Partial LCIs, which requires minimal user feedback. Parts from type (1) are removed during preprocessing, while parts from type (2) are what we usually encounter in a traditional LCA. To compare parts with partial

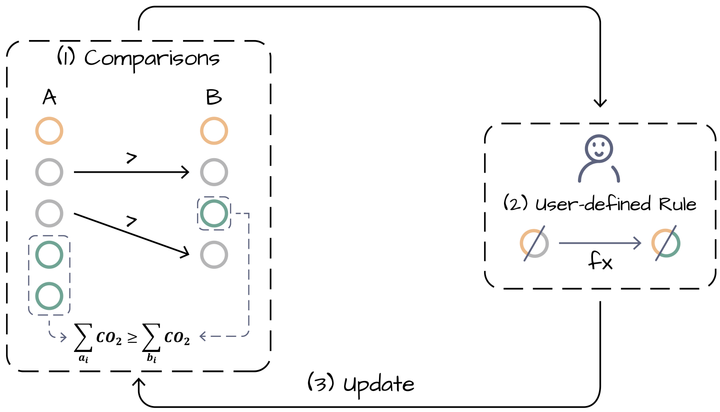


Figure 3.6: Our comparison algorithm uses parts either for pairwise comparisons (grey nodes) via heuristics (edges) or for carbon footprint comparison (green nodes). (1) The initial comparison results are first delivered to the user. (2) Drawing from the user's domain knowledge, the user adds comparison rules using any parts available to solve the unmatched parts (orange nodes). (3) The results can be updated, taking into account the user-defined rules.

information (3), we use expert-designed *heuristics*, which allow us to determine whether part a_i from $\mathcal{A}$ will have greater EI than part b_j from $\mathcal{B}$ without ever computing the actual EI of either part. Intuitively, these heuristics can be thought of as conservative rules to compare parts. For example, a heuristic may state that if two ICs are manufactured with the same processes but one is significantly larger than the other, the larger one will have greater EI. This means that the key algorithmic challenge for reasoning about parts with partial information is to perform a *pairwise matching* between two sets of parts using these heuristics. Importantly, this matching algorithm must also optimally leverage the EI information that we can obtain for parts from type (2). We tackle this challenge with a comparison algorithm which is explained in Section 3.2.3.

User-in-the-loop The output of the comparison algorithm is two design subsets $\mathcal{A}_\delta$ and $\mathcal{B}_\delta$ (nodes except for the orange ones in Figure 3.6) for which we can show that the EI of the former is bigger than the EI of the latter. For the remaining parts (orange nodes), i.e., $\mathcal{A} \setminus \mathcal{A}_\delta$ and $\mathcal{B} \setminus \mathcal{B}_\delta$, our algorithm either has insufficient information about the parts, or design $\mathcal{A}$ is simply not more environmentally impactful than design $\mathcal{B}$. In both cases, the user is presented with the comparison result and the remaining parts. To refine the assessment, they can then provide more information to the system by completing part information that the inventory phase has been unable to retrieve or by providing additional, user-defined rules about the relative impact between parts. Once more information has been provided, either about the parts themselves or about the relationship between parts, the comparison result can be updated. Our interactive feedback loop is illustrated in Figure 3.6: (1) the two designs are automatically compared, and the results are delivered to the user; (2) the user can provide additional information to refine the comparison; (3) the comparison result gets updated, taking into account the new information.

In summary, DeltaLCA automates the inventory phase and the assessment phase by leveraging domain-specific knowledge to extract as much relevant information as possible during the inventory phase and by reducing the uncertainty in a comparative conclusion through part comparisons based on domain-specific heuristics during the assessment phase. As opposed to the traditional LCA comparison, with DeltaLCA, the user is only needed at the end for the actual comparison, not throughout the entire process.

3.2.2 Automated Life Cycle Inventory

The first step in the DeltaLCA pipeline is to generate an LCI from a PCB design file. Traditional methods for LCI require time and expertise for two key reasons. First, the component models available in LCA software and databases do not have a one-to-one mapping to the parts on a PCB; this prevents automation. Second, data for many parts is unavailable.

Since our goal is to obtain as much information as we can for a comparative assessment, we carefully selected the LCA-relevant features that can be extracted from PCB design files, as well as open-access data sources and APIs. This abstraction allows us to minimally represent a PCB design loosely as a collection of components with different attributes (see Section 3.2.3).

To achieve the desired level of accuracy and automation, a four-step procedure is employed: (1) parsing the raw design files locally. This step aims to extract basic information about the components and board configuration within the PCB design relevant to LCA. (2) interfacing with external data sources to enrich our resulting parts list with up-to-date attributes of each component. (3) inferring core specifications, typically kept confidential by manufacturers, with all available information. These specifications, such as the die area of an IC, are crucial for assessing the component EIs. (4) estimating the carbon footprints of standard components with complete information.

3.2.3 Comparative Impact Assessment

The comparison algorithm takes as input the partial LCIs from the inventory phase for two designs $\mathcal{A}$ and $\mathcal{B}$. Without loss of generality, we assume that we want to show that the environmental impact, e.g., carbon footprint, of $\mathcal{A}$ is *bigger* than that of $\mathcal{B}$. In general, our algorithm will be able to show this for two subsets $\mathcal{A}_\delta$ and $\mathcal{B}_\delta$. Our primary goal is to design an algorithm where $\mathcal{B}_\delta$ is maximized (i.e., $\mathcal{B}_\delta \equiv \mathcal{B}$), thereby enhancing the likelihood of demonstrating that $\mathcal{B}$ has the lowest environmental impact with minimal or no additional user input.

As mentioned in Section 3.2.1, at this stage, partial LCIs contain parts with complete LCA information for which we can perform direct carbon footprint comparison and parts with only partial information which will be compared using pairwise heuristics. But how do we combine heuristic-based comparisons with carbon-footprint-based comparisons? One straightforward approach would be first to perform an LCA comparison for parts with complete information to calculate or approximate the carbon footprints, and then, in a second stage, perform a pairwise comparison based on heuristics. However, this could easily lead to selection problems. Suppose $\mathcal{A}$ only has parts with complete information and $\mathcal{B}$ only has parts with partial information. The first step would consume all parts of $\mathcal{A}$, even if not all of the per-part carbon footprints are necessary to weigh up against the carbon footprint of $\mathcal{B}$. Then, $\mathcal{A}$ would not have any parts left for the pairwise comparison. This example shows that the selection of parts used for the carbon footprint comparison and the pairwise, heuristic-based comparison are interdependent. We formulate this interdependent selection problem as an integer program, as described in Section 3.2.3. Lastly, we discuss our implementation and evaluate the performance in Section 3.2.3.

Heuristics

We define a *heuristic* **H** as a map from a subset of parts of design $\mathcal{A}$ and a subset of parts of design $\mathcal{B}$ to a decision about which subset has a higher carbon footprint, or if neither of them subsumes the other. The heuristic decision is based on certain part attributes, and we only assign heuristics between parts with the necessary attributes available. Heuristics are based on domain knowledge of the application domain and example heuristics to compare PCB components are listed below. We note that these can be extended and modified by the user based on available information:

1. **Package size for the same type of passive components.** For passive components, such as capacitors and resistors, those of identical dimensions consume comparable amounts of raw materials, making their environmental impact comparable, irrespective of variations in material composition.
2. **Chips with the same core and architecture.** Chips that share a common standard core and architecture (e.g., ARM Cortex-M), are considered equivalent in our analysis if they differ only in packaging type. This is underpinned by two factors: the environmental impact of the package is relatively overshadowed by the silicon part, and a shared core implies identical manufacturing processes and material usage, thereby equating their environmental footprints.
3. **Equivalent process.** In cases where components are similar in dimensions and weight, we assume that their transportation-related environmental costs are comparable. Furthermore, for components of the same category, the electricity cost in the assembly process, such as soldering, is consistent.
4. **Chip die size.** The impact of chip die sizes is two-fold. Larger dies not only suggest increased raw material consumption but also tend to result in lower production yields [22, 134], both contributing to a larger environmental impact.
5. **Process technology node.** More advanced (smaller in dimension) process technology node requires more steps and complexity in fabrication. This includes the need for advanced lithography equipment and multiple patterning processes, culminating in elevated energy consumption.
6. **Diode size.** Size is an effective indicator of raw material consumption across various diode types, such as photodiodes, LEDs, and Schottky diodes. Given their similar semiconductor structures, the size of a diode correlates directly with the environmental costs associated with the raw materials.

We use the first three heuristics to identify parts in $\mathcal{A}$ and $\mathcal{B}$ which are "identical," i.e., which parts we assume to amount to the same carbon footprint, given the partial information available to use. These parts are being removed before the comparison algorithm.

Note that we might encounter a situation where two heuristics evaluate to conflicting results. Two

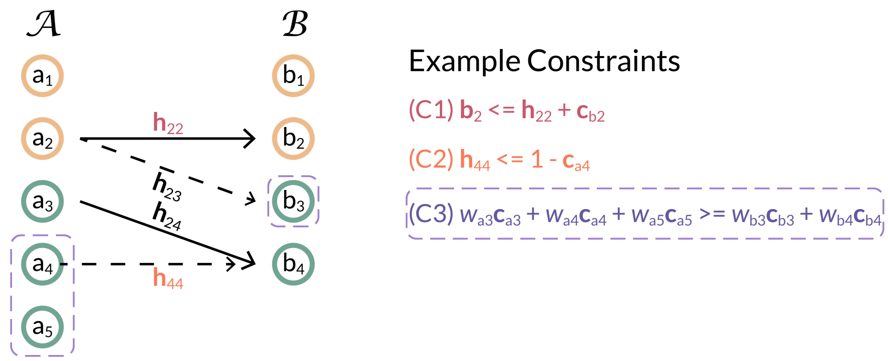


Figure 3.7: Input to this example problem is the bipartite graph, including the heuristic edges (solid lines represent selected heuristics and unselected are dashed) and the carbon footprint weights, see Table 3.1 for detailed values and variable assignments. The example constraints illustrate the three constraints from our integer program formulation.

Carbon footprints $\mathcal{A}$	Carbon footprint variables $\mathcal{A}$
$w_{a_1} = N/A$	$\mathbf{c}_{a_1} = 0$
$w_{a_2} = N/A$	$\mathbf{c}_{a_2} = 0$
$w_{a_3} = 10$	$\mathbf{c}_{a_3} = 0$
$w_{a_4} = 10$	$\mathbf{c}_{a_4} = 1$
$w_{a_5} = 10$	$\mathbf{c}_{a_5} = 1$

Carbon footprints $\mathcal{B}$	Carbon footprint variables $\mathcal{B}$	Part variables $\mathcal{B}$	Heuristic variables
$w_{b_1} = N/A$	$\mathbf{c}_{b_1} = 0$	$\mathbf{b}_1 = 0$	$\mathbf{h}_{22} = 1$
$w_{b_2} = N/A$	$\mathbf{c}_{b_2} = 0$	$\mathbf{b}_2 = 1$	$\mathbf{h}_{23} = 0$
$w_{b_3} = 10$	$\mathbf{c}_{b_3} = 1$	$\mathbf{b}_3 = 1$	$\mathbf{h}_{34} = 1$
$w_{b_4} = 10$	$\mathbf{c}_{b_4} = 0$	$\mathbf{b}_4 = 1$	$\mathbf{h}_{44} = 0$

Table 3.1: Carbon footprint values and variable assignments for the example problem from Figure 3.7. Parts for which we do not have complete LCA information have no available carbon footprint. The binary variables are assigned such that the objective function is maximized.

heuristic edges h_i and h_j conflict with each other if both consider the same subset of parts from $\mathcal{A}$ and the same subset of parts from $\mathcal{B}$ and if h_i and h_j make different decisions about which subset of parts has a higher carbon footprint. For example, if between a chip a_i and a chip b_j, a_i has a bigger die size (heuristic 4), but b_j has a smaller process node (heuristic 5), conflicting heuristic edges arise: heuristic 4 would conclude that $a_i > b_j$ while heuristic 5 would decide $b_j > a_i$. Because we cannot reason about them without more information, we remove all conflicting heuristic edges between parts before the comparison.

Comparison Algorithm

In this section, we explain the general integer program formulation for the comparison of two designs. Please refer to Figure 3.7 for a toy example.

Objective function The goal of the selection problem is to select parts of $\mathcal{A}$ and parts of $\mathcal{B}$ such that each part of $\mathcal{B}$ is shown to be of lesser carbon footprint than some part of $\mathcal{A}$, either through direct carbon footprint comparison or by selecting pairwise heuristics. More specifically, we formulate this selection problem as a binary integer program where we associate with each part b_i a binary variable $\mathbf{b}_i \in \{0, 1\}$. Here, $\mathbf{b}_i = 1$

means that part b_i has been included in the LCA comparison. The objective function is shown in Eq. 3.1. We are maximizing the number of compared parts of $\mathcal{B}$ since we want to do a comparison for as many parts as possible of that design.

$$\max \sum_i \mathbf{b}_i \tag{3.1}$$

Constraints $\mathbf{b}_i$ should only be put to 1 if it has been taken into account by a pairwise heuristic comparison or by means of its carbon footprint, which we translate via the constraint in Eq. 3.2.

$$(\mathbf{C1}) : \mathbf{b}_i \leq \left(\sum_{\mathbf{h} \in \mathcal{H}_{\mathbf{b}_i}} \mathbf{h} \right) + \mathbf{c}_{\mathbf{b}_i} \tag{3.2}$$

where $\mathcal{H}_{\mathbf{b}_i}$ is the set of all binary heuristic variables for heuristics which show that part $\mathcal{B}_i$ has been subsumed by some part of $\mathcal{A}$ and $\mathbf{c}_{\mathbf{b}_i}$ is a binary variable which is equal to 1 if $\mathcal{B}_i$ has been used in the carbon footprint computation. In the toy example in Figure 3.7, $\mathbf{b}_2$ has been successfully put to 1 because the heuristic h_{22} has been selected and b_2's carbon footprint has not been used.

As mentioned earlier, each part $\mathcal{A}_i$ of $\mathcal{A}$ can only be used in one heuristic. But if an $\mathcal{A}_i$ has already been used in the carbon footprint computation, it cannot be used for pairwise heuristic comparison. This is enforced by the constraint in Eq. 3.3

$$(\mathbf{C2}) : \sum_{\mathbf{h} \in \mathcal{H}_{\mathbf{a}_i}} \mathbf{h} \leq 1 - \mathbf{c}_{\mathbf{a}_i} \tag{3.3}$$

where $\mathcal{H}_{\mathbf{a}_i}$ is the set of all binary heuristic variables for heuristics which use part $\mathcal{A}_i$ and $\mathbf{c}_{\mathbf{a}_i}$ is a binary variable which is equal to 1 if $\mathcal{A}_i$ has been used in the carbon footprint computation. Intuitively, if $\mathbf{c}_{\mathbf{a}_i}$ is equal to 1, then the right-hand side of the constraint is equal to 0 and no heuristic for $\mathcal{A}_i$ can be selected. If the carbon footprint of $\mathcal{A}_i$ has not been used for comparison, then $\mathbf{c}_{\mathbf{a}_i}$ is equal to 0 and the right-hand side is equal to 1, which specifies the constraint that at most one heuristic for $\mathcal{A}_i$ can be selected. The effect of this constraint can be observed in Figure 3.7, where $\mathbf{h}_{44}$ could not have been selected because $\mathbf{c}_{a_4}$ has already been set to 1.

Finally, we handle parts selected for carbon footprint comparison. For each part with complete information, we assign its carbon footprint as its weight w_i. To show that the carbon footprint of $\mathcal{A}$ is bigger than that of $\mathcal{B}$, we must ensure the weighted sum of the parts selected for carbon footprint comparison from $\mathcal{A}$ is bigger, which is expressed via the constraint in Eq. 3.4:

$$(\mathbf{C3}) : \sum w_{\mathbf{a}_i} \mathbf{c}_{\mathbf{a}_i} \geq \sum w_{\mathbf{b}_i} \mathbf{c}_{\mathbf{b}_i}. \tag{3.4}$$

In the toy example in Figure 3.7, the left-hand side of this constraint sums up to 20, which is greater than the right-hand side which selects only $w_{b_3} = 10$.

Summary The objective of the integer program is to maximize the sum of $\mathbf{b}_i$ (Eq. 3.1). However, for $\mathbf{b}_i$ to be 1, either a corresponding heuristic variable $\mathbf{h}$ or its carbon footprint variable $\mathbf{c}_{\mathbf{b}_i}$ have to be assigned to 1 (Eq. 3.2). A heuristic can only be selected if the counterpart in $\mathcal{A}$ has not already been used by another heuristic or for direct carbon footprint comparison, and we ensure that resources in $\mathcal{A}$ are only used once (Eq. 3.3). The carbon footprint comparison is only valid if we have enough carbon footprint in $\mathcal{A}$ to counterbalance that in $\mathcal{B}$ (Eq. 3.4).

Note that theoretically, a part $\mathcal{B}_i$ could be shown to be subsumed by $\mathcal{A}$ via multiple heuristic comparisons and/or carbon footprint comparison at the same time, i.e., we do not enforce comparison uniqueness as for $\mathcal{A}_i$ in Eq. 3.3. However, it is not advantageous for the solver to put both kinds of variables for $\mathcal{B}_i$ to 1 since both would have to be weighed up against by parts in $\mathcal{A}$ and we can only use parts in $\mathcal{A}$ once.

Implementation

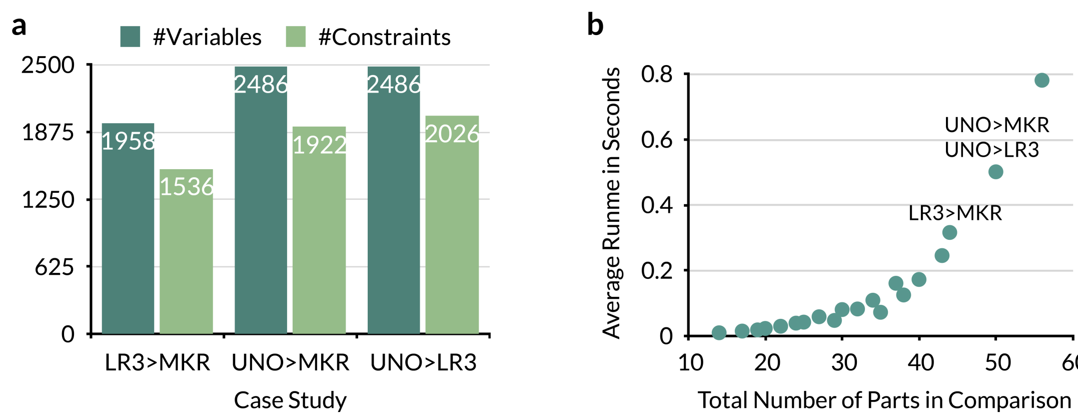


Figure 3.8: (a) The size of the integer program illustrated by the number of variables and the number of constraints for each comparison in the case study (Section 3.2.4). (b) Average runtime (on a MacBook Pro with M1 chip and 8GB RAM) in seconds over the problem size (represented by the total number of parts involved in the comparison). Here "parts" refers to the ICs in the design and two special parts generated by the automated life-cycle inventory process: passive components (such as resistors, capacitors, and inductors) and board substrates are precomputed as two single parts.

Integer Program We implemented the integer program using the Python API of OR-tools [71] and for common comparisons, the runtime is a fraction of a second. Although integer programming is NP-complete and combinatorial in nature, in practice, solutions to a problem of a reasonable size can be found efficiently. Since for each part of $\mathcal{A}$, we introduce heuristics with all compatible parts of $\mathcal{B}$, the number of heuristic variables scales quadratically w.r.t. the problem size, i.e., the number of parts. This means that the runtime also scales approximately quadratically w.r.t. to the number of parts. We evaluated the runtime of our comparison algorithm by making pairwise comparisons among eight PCB boards (the three complex designs from the case study in Section 3.2.4 and five smaller designs). As shown in Figure 3.8, our comparison algorithm's average runtime for a problem of a size similar to the complex design comparisons (i.e., 40 ~ 50 parts in comparison) is around 0.5 seconds and thus enables our UI to run at interactive speeds.

Looking beyond the specific examples above, we also observe that more complex designs with high part counts consist primarily of passive components such as resistors and capacitors that are simplified by DeltaLCA. For example, a sample laptop motherboard design has 443 parts, of which 353 (80%) are passives [140]. Similarly, the main electronics board for a Roomba contains a total of 633 parts, of which 79% are resistors or capacitors [122]. Additionally, many devices like smartphones and tablets consist of multiple smaller PCB modules [49]. Our approach enables rapid analysis of subsystems and thus the evaluation of complex assemblies of parts.

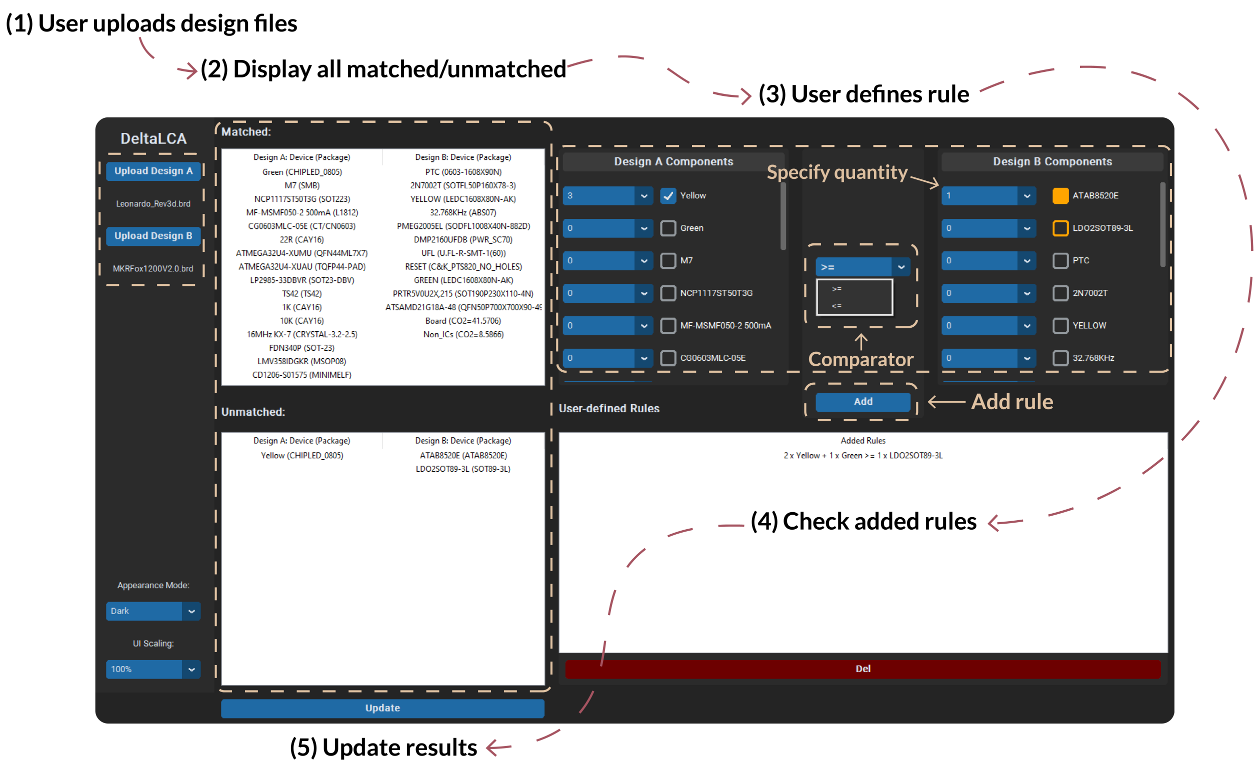


Figure 3.9: System Screenshot of the DeltaLCA user interface. (1) The user uploads design files using the buttons in the sidebar. (2) The user is presented with matched/unmatched components. (3) The user specifies custom comparison heuristics. (4) The user checks all added rules. (5) The tool updates the comparison results taking into account user-defined heuristics.

User Interface for Comparing Designs The UI for DeltaLCA is developed with Python's tkinter framework (shown in Figure 3.9) and bridges the automated LCI with the comparison algorithm. After the user uploads two design files using the buttons in the sidebar, the system initiates an inventory generation process. The inventory may be partially complete due to the incomplete component information from the design file.

Then the initial comparison runs automatically and the user is presented with two tables: one displaying the matched components from designs A and B respectively, and another below listing any unmatched components. The user can analyze individual components in each design via a scrollable frame on the top right. All unmatched components are highlighted in yellow and located at the top of the list to emphasize their selection priority, in contrast to matched components shown in gray. The user can create custom comparison heuristics by selecting components and specifying their quantities via checkboxes and dropdown menus before the component names, and setting the comparison direction through the comparator option menu. Custom user-defined rules are added to a separate table using the “Add” button, while the “Del” button allows for the removal of specific rules. The user can then click the “Update” button to re-run the comparison algorithm taking into account their custom rules. The comparison process concludes once all components from the design deemed to have a lower EI are successfully matched.

3.2.4 Case Study Evaluation

In the realm of electronics, most designs are rather complex; they incorporate a diverse range of components and the components often have incomplete information. To understand DeltaLCA’s capability in comparing complex PCB designs with incomplete information, and whether users could use the DeltaLCA UI to interactively arrive at a comparative conclusion, we did a case study that compared three Arduino dev boards: (1) Arduino Leonardo R3, (2) Arduino MKR FOX 1200, and (3) Arduino UNO Wifi. We choose these devices for several reasons: (1) Complexity: Unlike simpler devices like a mouse, these development boards are akin to small computers, featuring intricate circuitry and multiple-layer design. (2) Relevance in UbiComp: these platforms are widely used for prototyping and educational purposes.

Our study involved a cross-comparison of these three devices. One expert with five years of electronics design experience, but no LCA experience, used our tool to establish a relative EI ranking of the three boards, namely $(3) \geq (1) \geq (2)$. For reference, we also collected some basic statistics about the three boards in Table A.1. If we only deduce the relative ordering based on the information in this table, it is not obvious what the relative order should be. In fact, the expert user initially tried to prove $(2) \geq (1)$ with our UI but in the end could not figure out a way to do so and realized that it should be the opposite direction. This demonstrates that our tool could also be used to help users verify their hypothesis about device EI comparisons.

It is worth noting that although the comparison results are consistent within the pairs of the three boards, we do not have full LCA results for these devices as full LCAs are still rarely available for many electronic devices. But DeltaLCA’s capability to directly compute simpler PCBs’ carbon footprints given full information (see Section 7.1.1 of Zhang et al. [218]) does serve as evidence that DeltaLCA can achieve relative consistency within paired comparisons.

As shown in Figure 3.10, for complex devices like these, containing around 100 components and 20~30 ICs, our tool’s comparative algorithm can automatically match most components (on average 88% overall, and 94% for the target, design B in this case) without any user intervention.

Figure A.1 shows that typically fewer than five user-defined rules (specifically, 4, 2, and 1 for the three

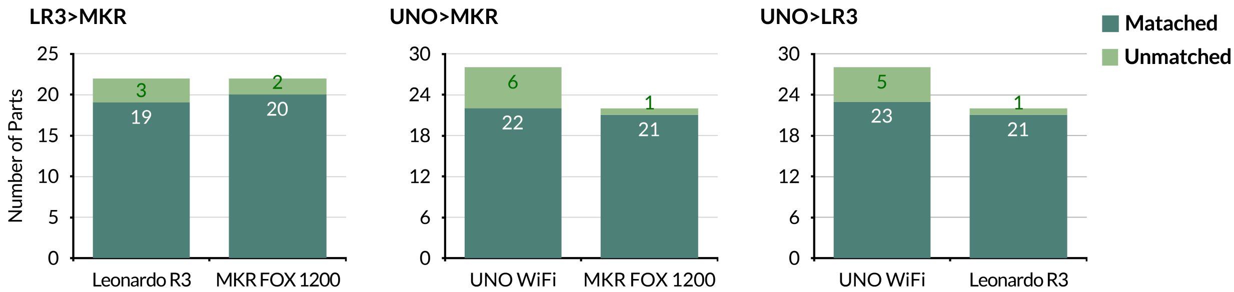


Figure 3.10: The matching coverage of our comparison algorithm without any user-defined heuristics for the three cases. Our method can automatically find a valid matching that covers most of the parts in both designs. Here the parts being counted refer to the list generated by the automated life-cycle inventory process where we turn components like resistors, capacitors, and inductors into a single component "NonICs" and the base board itself into a single component "Board".

comparisons) are needed to complete a comparison, with an average time spent of less than 15 minutes per comparison. This is a significant reduction in time needed compared to traditional LCAs for simpler designs (mouse example in Section 7.1.1 of Zhang et al. [218]). Additionally, our tool does not require LCA-specific expertise, empowering electronics designers to perform evaluations directly. Such efficiency demonstrates the substantial time savings DeltaLCA offers designers.

In our study on complex designs with partial inventories, we highlight the strengths of our comparison algorithm which can utilize both carbon footprint data from automated LCI when available, and also apply expert-defined heuristics when only partial information exists. Moreover, users have the flexibility to create rules for any components across the designs, not just the unmatched ones. This feature is crucial because the matching derived from the integer program may not always be globally optimal and users might have insights on specific components that are not automatically extracted from PCB design files and online data sources.

3.2.5 Discussions and Conclusion

In this chapter, we introduce an innovative method for creating design tools that facilitate environmental-impact-aware decision-making. This approach is grounded in the key understanding that, although LCA calculations are complex, comparative analysis can enable effective decision-making even with incomplete data. Our end-to-end tool, specifically developed for comparing carbon footprints of electronics designs, demonstrates that leveraging domain-specific knowledge and datasets can significantly automate the process and require minimal user input. We evaluated our tool's effectiveness through case studies and a user survey.

This work presents a new approach to evaluating the environmental impact of electronics and designing for sustainability that opens multiple directions for future research. First, there is considerable potential to scale the system to more complex devices and with different inputs. For example, DeltaLCA currently requires the full design files to compare devices. Future work could explore ways to leverage our heuristic matching approach to perform comparisons with higher-level information such as the subcomponents within a device (e.g., display, camera module, battery). However, scaling our tool to accommodate this greater complexity of components will require additional engineering efforts. Future research could improve the Automated LCI to be compatible with a broader range of electronic components beyond passive components and ICs such as display sub-assemblies like cameras and radio modules. Potential data sources include publicly available specifications of the products with their size and power consumption as well as data from public product environmental reports of similar devices that could be used to develop heuristic relationships.

Second, improvements to Automated LCI could be achieved by employing novel search algorithms, with advancements in Large Language Models (LLMs) offering possibilities to explore broader databases or to identify freely available web data. This approach could increase automation by augmenting our heuristic matching algorithm that seeks to minimize unmatched components and reduce the number of additional user rules required for making final decisions. Currently, the number of matches is used as a proxy for this goal, but future work may reveal more effective methods, possibly through deeper domain expertise or advanced data analysis.

Third, our approach could be extended to perform comparative assessments at different stages of the design pipeline. Although our current research focuses on comparing two valid PCB design files, the concept of early-stage or abstract comparisons, such as a list of components or even a written paragraph describing the fabrication process of a device, is a promising area for future work. For example, comparing the environmental trade-offs between high-level system architectures could provide valuable information at early design stages. Additionally, zero-knowledge comparisons where design files are not accessible, could provide greater insights into Scope 3 emissions such as components from suppliers that do not provide environmental impact data. Such endeavors would likely require not only direct heuristic comparison but also probabilistic reasoning to model uncertainty in the data and constraints. Future research could also adapt our insights on comparative approaches to LCA to other domains, such as architecture, material synthesis, and garment production.

Chapter 4

Design with Sustainability Constraints

In this chapter, we show how we can create design systems that consider different kinds of sustainability constraints. First, we introduce a design system for a fabrication process that is itself material-efficient and more sustainable in Section 4.1. This work takes into account constraints from a more sustainable manufacturing process, focusing on how we can represent objects that are provably fabricable with that process while making it easy for users to design with this representation. Second, we introduce a design system that generates designs solely from existing objects in Section 4.2. The constraints now come from the materials and components used to fabricate the designs, and similarly, we emphasize how to represent designs made from existing objects while allowing users to explore the design space easily.

4.1 Case Study: Design for Material-Efficient Fabrication Processes

4.1.1 Introduction

Knitting is a versatile craft with rich aesthetic and functional design spaces. Its scope ranges from garments and toys to architectural structures and medical implants. The ubiquity of knit textiles in our lives is driven by programmable knitting machines. Machine knitting has the potential to become the next 3D printing: knit textiles are pervasive, customization of knit objects like clothing is valued, and machine prices have fallen within reach of maker spaces, small shops, and hobbyists.

Knitting is also in itself an environmentally friendly way of fabricating textile products. Compared to cut-and-sew, which generates fabric leftovers that cannot be easily reused, we can knit only the needed panels for a design, avoiding wasting materials. In addition, knit items can be unraveled and in a sense the same materials can be repurposed in many other ways.

However, makers lack design tools that provide needed control over familiar design axes, enable customization of existing designs, and encourage exploration of the design space. Consider the variety of dress shapes in Figure 4.1. There are several options for knitting patterns that can construct these shapes. The knitting designer must choose from patterns like these to achieve functional and aesthetic effects. Based on

Figure 4.1: Our interactive design system helps users explore key design axes for knitting to generate highly customized patterns from input shape templates; e.g., a seamless yoke dress with princess-cut apparent seams (a), and drop shoulder dresses with textures on the arms and skirt (b-d). The output of our system is a knit pattern template that lets users vary the shape while preserving the design, for example, creating a child's dress with short sleeves (d) that matches an adult dress (b), or varying skirt texture and angle, and sleeve knitting direction (c). The system guarantees that all results and variations are machine knittable.

interviews we conducted with knitting designers, we identified seven design axes that are typically present in knitting patterns.

Shape Variability. Specific aspects (or geometry parameters) of a knit object's shape and size can vary without requiring a wholly new pattern. For example, a pattern designer might vary the length of a skirt and sleeves and the height of the waist to change an adult's dress to a child's dress (**b** vs **d**).

Composition. The basic building blocks of knit objects are sheets and tubes, which are composed to form a shape. For example, shoulder design in garments may be composed as smoothly merging or abutting tubes (**a** vs **b**).

Seaming. Related to composition, seams are used to connect building blocks (e.g., the shoulders in **b**) but also within a building block (e.g, the sleeve in **c**).

Orientation. Knitting looks different, and stretches differently, in the horizontal and vertical directions. As the sleeve of **c** shows, changing orientation can change the locations of seams. Thus, the orientation, or alignment of stitches along the surface, is an important design choice.

Surface Layout. Knitted objects are typically comprised of conceptually meaningful regions. Layout includes the axes of symmetry, a line of increases or decreases, and the boundaries of a texture region, such as textures on the dress skirts, and the apparent seams in (**a** and **c**).

Curvature Shaping. Sheets and tubes are flat grids of stitches until shaping stitches are added, creating non-grid formations that add intrinsic curvature. Curvature is distributed around the bodice in (**c**) and concentrated towards the front in (**d**).

Surface Texture. Texture is achieved by varying stitches on the surface of a sheet or tube without varying curvature. Complex surface patterning of stitch variants gives knitting its aesthetic versatility, as shown in the dress skirts (**a-d**).

Customization and exploration of designs require *interactive control* of these seven design axes. There are two challenges to interactive design. First, knitting patterns must meet several discrete local and global constraints in order to be fabricable. Stitches must cover the surface with a small number of yarns while avoiding helical structures that cause cyclic dependencies in the fabrication process. Shaping stitches must be placed to capture mesh curvature, but also respect limitations on their type and relative alignment. Second, the seven axes are strongly intertwined at the stitch level, so making a decision along one axis can undo decisions made along other axes. For example, changing the *orientation* of stitches in the sleeve of the dress between (**b**) and (**c**) requires different *curvature shaping* stitches, changes the *composition* of the sleeve from a tube to a sheet, requiring a *seam* down the length of the arm, and rotates the *surface texture* by 90 degrees. Ideally, a design tool should enforce the constraints without overly limiting the designers' ability to explore, which is hampered if design decisions undo each other.

Prior work addressed several of these design axes. For example, Yuksel et al. [213] demonstrate that a coarse quad-dominant mesh modeling of geometry enables the representation of important design axes of knitting, such as orientation and surface texture, and supports iterative modifications to these axes. However, without a design tool that can *automatically generate the quad mesh from high-level design input*, this representation cannot support iteration on axes such as composition and shape variability. Similarly, without a *strong theoretical connection between knittability constraints and the algorithm that generates a pattern from a quad mesh*, it is impossible to guarantee knittability. As a result, a design tool cannot suggest design solutions or warn users when they make changes that will break their design.

To address these limitations, we introduce two theoretically-grounded advances. First, we present a **novel meshing tool** that expresses the theoretical relationship between the singularity structure of quad meshes and the knitting design axes. This lets users of all knitting design skill levels generate coarse meshes that satisfy their design goals and avoid helical structures that lead to undesirable patterns. Further, our algorithm takes as input a parametric template geometry that can vary over a specified parameter space, for example, a dress whose sleeves can vary from short to long or whose skirt can be elongated, allowing a design to be customized for a user.

Second, we introduce **formal knittability criteria** over the coarse mesh to ensure knittability without over-constraining the design space. By *knittability* in this work, we mean a valid *machine* knitting patterns in conjunction with constraints to account for physical limits of knitting machines and yarn that improve design robustness. Our validation and accompanying algorithms enable not only notifications about knittability problems, but promote interactive design across multiple design axes supported by auto-completion and automated design suggestions.

Based on these theoretical insights, we contribute a practical knitting design framework that supports:

- Requirements for design axes drawn from real-world knitting design experts;

- Variable *template* patterns that correctly propagate design decisions as parameters of the geometry are modified;
- Direct manipulation by users of multiple, interdependent design axes;
- Automatic knittability checking and auto-complete assisted design; and
- Generation of knitting machine instructions.

4.1.2 Understanding Knitting Design and Its Relation to Quad Meshes

In this section, we introduce some background knowledge on knitting and quad meshing. Knitting builds on a long craft, design, and artistic history [174]. Knitters can refer to books (e.g., [21]) or websites (e.g., Ravelry [154]) that include a wealth of knitting patterns and design strategies. We first briefly review how knit objects are constructed (see [187, 127] for a more thorough review). Then, we present the key results of our study with knitting pattern designers, which motivated our design representation. Lastly, we explain how knitting relates to quad meshing, an insight that enables knitters to design using our tool without knowledge on geometry processing.

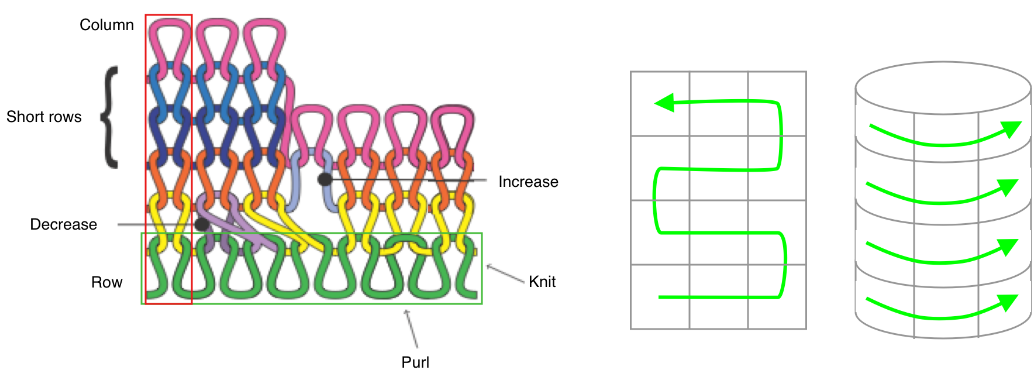


Figure 4.2: Knit object construction. (left) A segment of knit fabric showing the basic stitch types and curvature shaping. (right). The two basic knitting primitives: sheets and tubes. Grid cells represent stitches, and arrows denote order of fabrication.

Constructing Knit Objects Knit objects consist of a grid-like fabric of interconnected stitches. A grid of stitches can be a *sheet* of fabric, or the ends can be joined to form *tubes*, called *knitting in the round* (Figure 4.2). Knit objects are composed by joining and cutting these elements in various ways and orientations.

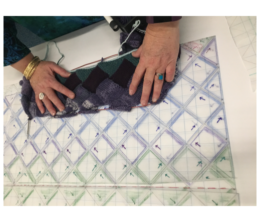
Stitches in the grid are formed sequentially by pulling a yarn loop through a "parent" loop in the row below it. To create a stitch, the loop it is pulled through must have been knitted already. This means that knitting inherently constrains the order in which stitches can be constructed. The central constraint of knitting is that the graph of stitch dependencies must be non-cyclic. The first row of stitches have no parent stitches, so they must be created with a special type of stitch called a *cast-on*. Similarly, the final row of stitches is closed with a stitch called a *bind-off* which acts to stabilize the knit object. This is important because a non-bind-off stitch with no children can unravel.

The grid can be locally distorted by adding and removing loops to add curvature to the fabric. This is done along columns using special *increase* and *decrease* stitches and along rows using *"short rows"* (Figure 4.2). Also, by varying the direction each loop is pulled through (knit and purl) – as well as other loop properties that create twists, holes, and overlayed loops – it is possible to create *surface texture*, such as cables, lace, and ribbing. As an example, varying the order that loops are stacked or pulled in an increase or decrease can create *leaning* increases or decreases, which appear to tilt left or right. Aligning several leaning stitches in a line can create the appearance of a seam in the fabric, as visible in Figure 4.8 (b). Throughout the paper we will use "apparent seam" to distinguish these seam-like stylistic choices from true seams, which occur when separately knitted edges are sewn together as a post-process.

Knitting machines do not change the important axes of knit design, but they do add fabrication constraints not found in hand knitting. A V-bed knitting machine contains two beds of small needles at fixed spacing. A piece of yarn is shuttled back and forth between the beds by a carrier, and the needles are programmed to interact with the yarn (e.g., grabbing a loop), or each other (e.g., passing loops) as the yarn passes. Each needle holds one or more active loops at a time, and only these loops can be built upon. Once a machine drops a loop, it cannot pick it back up, so all stitches needed in the future must be held on needles. While loops can move between needles to create gaps for increases and overlaps for decreases, the physical dimensions of needle size and spacing limit how large a gap can be without snapping the yarn (*overstretching* the yarn), and how many loops can overlap before dropping off the small needles (*overstacking a needle*). These impose an upper bound on the number of loops each stitch can increase or decrease, which we conservatively cap at two, and make short-rows preferable for shaping (for hand knitters, increases and decreases are preferred).

Designing Knitting Patterns: A Research Survey and Study To understand how designers construct knitting patterns, we surveyed popular keywords used with over 200,000 free knitting patterns available on Ravelry [154] and conducted a contextual inquiry [10] with five knitters, focusing on the design process, motivations behind design decisions, and the use of patterns and other artifacts or tools.

Our *survey* of patterns showed that the most popular search keywords specified composition (129K patterns mentioned seaming; 160K mentioned seamless; almost all specified sheets (flat, 227K patterns) or tubes (in the round, 181k patterns)). Next in frequency came orientation (163K patterns) and shaping (32K patterns use short rows, which could underestimate the importance of shaping since almost all patterns use increases and decreases). The use of such keywords suggests that the identified design axes are of interest not only to pattern designers but also to knitters.



Regarding *study* results, participants tended to enter initial planning stages based on some inspiration (e.g., a picture) or an internal image of the final object they wanted to create. They drew this out as a sketch of the objects' composition or directly translated it onto a grid (a stitch-level representation) using perler beads or graph paper (see inset figure). They also determined knitting orientation, texturing, added symmetry, and created an assembly plan, as needed, at this phase of the project. More complex objects were broken down into different components to be designed individually.

Participants also discussed the challenges of modifying an existing pattern by resizing, coloring, texturing, and modifying design elements. Among these, resizing was by far the most common since small changes (like the specific knitter, yarn, and needles) could alter the number of stitches needed to achieve their goal. They did not have automated methods to do this. Instead, they used arithmetic, visual inspection, trying things on, or comparing theirs to a to-scale pattern. Because many knitters preferred to knit in the round, a second common change was to the composition of a pattern of sewn sheets (such as a sweater) into a tube. Thus, a knitting design tool could be of value not only to designers but also to the much larger group of knitters who simply want to make things that fit. This also demonstrates that stitch level decisions can be deferred until fabrication time while still respecting a design intent, indicating that the design is actually captured by a higher-level structure.

Our survey and study provide evidence that an ideal tool for knitting pattern design should support changes in the target shape (especially size), composition, orientation, seaming, curvature shaping, texture, and surface layout features such as symmetry. However, simply supporting these design axes is insufficient. Knitters want to modify them, which is currently time-consuming and difficult. They struggle to ensure that the resulting pattern will be knittable and to preserve one design decision when modifying others.

Quad Meshing and Knit Design Quad meshes are meshes with only quadrilateral faces, and the mesh vertices typically have 4 adjacent edges (or 3 if on boundary). Such vertices are called regular, while others are called irregular or singular. The surplus or deficit of adjacent edges is the index of a singularity (a singular vertex). A typical knit stitch, like a quad in a quad mesh, has exactly four neighboring stitches, a compelling parallel that motivates the most popular stitch-level representation: Stitch Meshes [213]. In addition to this low-level correspondence, we found two higher level correspondences.

First, based on our study with knitters, we see that they think and lay out knitting designs in sheets and tubes. When they combine the sheets and tubes, these structures may come together in greater or fewer than 4 edges, creating singularities. We found connections between several common knitting patterns and groups of singularities with particular indices, which we compiled into a set of composition rules that can be applied to a surface to control how the surface is broken down into sheets and tubes.

Second, knitting has two orthogonal directions formed by rows and columns of stitches (course and wale, respectively). In quad meshing, locally orthogonal axes are represented by a *cross field* – a pair of vector fields over a surface that are always locally orthogonal. Several methods for converting a triangle mesh to a quad mesh (remeshing) use cross fields to guide the quad orientation. The orientation of rows and columns in knitting is an important design decision, so we employ a cross field to capture the user's design intent.

Quad meshes have rows and columns found by following neighboring quads on opposite edges. If two different quads in the same row are also in the same column – meaning there is a cycle, the row is a helix. While knitting in the round is technically knitting one helix, in our representation (and also in standard knitting patterns), this helix is not explicitly represented. Instead, it is broken into individual rows, and the overall helix is constructed only at knitting time when transitioning between these rows. This view of knitting makes it easier for us to reason about stitch construction dependencies. Specifically, all of the stitches in a row must be constructed before any stitches in later rows. A quad mesh helix creates a cyclic dependency between stitches, which is not knittable.

4.1.3 Related Work

Knitting design research can be segmented into three domains of inquiry: *representation*, which is typically stitch, primitive or mesh-based; *pattern knittability*, which includes both generation and verification of hand and machine knit patterns; and *interactivity*, which includes support for the seven design axes identified in our survey and study. In addition to these topics, we will discuss prior works on quad meshing for knitting, an important step in our system.

Representation of Knit Patterns Representations fall into three categories. Stitch-based representations specify individual knitting operations and can be written as language, charts, or annotated meshes. Primitive-based ones address tubes and sheets directly, while patch-based ones extend mesh representations to multiple stitches per element.

Traditionally, knitting patterns are conveyed as stitch-based fabrication instructions, typically in a language called *knitspeak* [81] or visually in a chart (e.g., [19, 167, 179]). Several systems use quad-dominant meshes rather than a chart or language to represent objects at the stitch level [87, 213, 203, 204]. In these systems, quad faces represent regular stitches, triangles the ends of short rows, and pentagons increases and decreases. Additional data is embedded to indicate orientation and to differentiate stitch types, such as knits from purls. While stitches directly correspond to the fabrication process, knittability is a primary issue here, which we discuss in the following section.

Since knitting is composed of tubes and sheets, an intuitive alternative is to specify a knit object as a composition of parametric sheet and tube primitives (e.g., [127, 102]). This approach has the advantage of supporting parameterization of these primitives, enabling a single pattern to act as a template for customization. It is also possible to provide knittability guarantees over this representation, though matching an arbitrary input target shape is not straightforward.

If we generalize meshing to represent multiple stitches per quad, we gain many of the benefits of primitive-based approaches, while still being able to match an input target shape [213]. However, this approach has not been extended to include pattern generation and ensure knittability, and depends on a high-quality patch input that aligns with the desired design.

Knittability Knitting an object requires the generation of a valid sequence of stitches. Because stitches are created by pulling loops through other stitches, such an ordering is not guaranteed to exist. Thus, stitch-based representations typically lack inherent guarantees of knittability (though commercial systems warn about potential failures). When representing the dual graph of a stitch-level mesh, dependency errors in a pattern can

	Variable Shape	Compositon	Orientation	Surface layout	Seaming	Curvature shaping	Surface texture	Machine Knittability	Hand Knittability	Mesh Input
Igarashi et al. 2008		☆							★	★
Yuksel et al. 2012		☆	☆	☆		★	★			☆
Wu et al. 2018			☆							★
Wu et al. 2019			☆	☆	★	★	★		★	☆
Popescu et al. 2017								★	★	☆
Narayanan et al. 2018			☆			☆		★	★	★
Narayanan et al. 2019			☆	☆		★	★	★	★	★
McCann et al. 2016	★	★	☆			☆		★	★	
Kaspar et al. 2019	★	★	☆			☆	★	★	★	
SHIMA SEIKI/STOLL		☆	☆	☆		☆	☆			
Our Work	★	★	★	★	★	★	★	★	★	★
	Direct Interactive Control							Desirable Features		

Table 4.1: Desirable features of a knitting design tool supported by literature. Unfilled stars show partial satisfaction of the goal; either users have direct, but incomplete control, or complete but indirect control. For example, Yuksel et al. [213] gives direct, but incomplete control over orientation since orientations may only align with coarse input mesh edges, and Narayanan et al. [137] gives complete but indirect control over surface layout since symmetries and feature placement can be specified exactly by moving stitches one by one. If properties are assumed from the input, or are both indirect and incomplete, they are not considered controlled. An unfilled star for mesh input indicates that pre-processing outside the system is required.

be found by cycle checking. Popescu et al. [152] pioneered the graph approach with a hybrid representation, and Narayanan et al. [136] extended it by directly constructing a graph over an input model and identifying a sufficient set of constraints on the graph to guarantee machine knittability. However, these checks must be done for every iteration to a pattern.

Using a primitive-based representation, McCann et al. [127] developed a provably correct transfer planning algorithm that schedules a pattern on a machine knitting machine. Kaspar et al. [102] extended this work with more primitives, more composition options, and a more robust texturing system.

Knittability guarantees have not been demonstrated for patch-based representations. This would require imposing constraints that ensure knittability. If it were possible to guarantee knittability, a patch-based representation would be preferable to stitch-based approaches because of its generality, and to primitive-based approaches because of its flexibility in easily representing a wide variety of input shapes.

Interactive Knit Design Tools Stitch-level mesh-based approaches are intuitive for representing underlying geometry, can automatically generate a knittable solution for a specific geometry, and support low-level control over specific stitches. For example, Narayanan et al. [137] support direct stitch mesh editing while ensuring machine knittability. However, concepts such as composition, orientation, and surface layout are not directly represented in a stitch-based mesh; rather, they are expressed through the stitches that are specified. Thus, the design tool cannot know when they are violated. Further, all these modifications are lost if the original object's geometry is changed.

In contrast, primitive-based methods guarantee machine knittability and allow shape to be varied parametrically since primitives can be parameterized [127, 102]. Such methods defer stitch-level decisions until instruction generation, which allows interactive editing of composition and other design goals, such as curvature and texture (see Table 4.1). However, this approach has two key disadvantages. First, it requires expertise to model a desired shape, making it particularly challenging for applications involving more complex geometry. Second, while it is possible to control composition and shape variations, editing is indirect, requiring expertise to achieve even simple variations that can depend on multiple parameters in a complex way—a classic problem in parametric CAD systems [211].

Supporting direct editing of design goals requires a representation that relates stitches to shapes. Patch-based approaches have this potential. Prior work demonstrated the power of patches to allow control over curvature shaping and surface texture, and to allow movement and changing of stitch types [213]. However, several key limitations remain, which our work addresses. First, Yuksel et al. [213] use a coarse polygonal mesh as input, which requires a high level of user expertise to generate, and limits interactive control over composition and orientation. Our first key contribution shows that by developing meshes that correspond to a knitter's conceptual breakdown of a knit object, we can enable control over multiple important design axes. In particular, we prove that by controlling singularities in the mesh, knitters can intuitively and directly specify and iterate on these design axes and generate helix-free quad meshes that are necessary for knittability.

Further, Yuksel et al. [213] are not concerned with knittability, generating patterns only suitable for simulation and rendering. While rules have been developed to ensure machine knittability on stitch-level meshes, where constraints come directly from analyzing the fabrication process [136], it is not trivial to extend this to patches. This is the second key contribution of our work. We design a lightweight set of high-level patch constraints that don't over-constrain the design space but enable us to create and formally prove the correctness of, an algorithm for translating them into knittable patterns. This allows novel system-level contributions: constraints can be directly encoded in solvers, enabling interactive verification and completion during labeling, automatic seam placement, and geometric optimization, while respecting machine constraints and shaping preferences.

Quad Meshing Our method works at the patch level, which is defined as a coarse quad mesh on the input surface. Quad meshing is an active research area and we refer readers to [15] for a survey. The fundamental challenge in applying existing quad-meshing techniques to patch-level knitting design is allowing users to control the composition and surface patch layout. Extensive work on quad meshing [15] has shown that field-guided methods best enable user control. In field-guided methods, orthogonal vector "cross"-fields on the surface are optimized for a given smoothness energy and to meet user specifications (e.g., direction strokes). A quad mesh is then created by finding a parameterization whose gradients are optimally aligned with the field. For a review of concepts in field design, we refer interested readers to Section 3.2 of a state-of-the-art report [193].

Despite great advances in this area, directional control while avoiding helices remains challenging, particularly for coarse meshes. Solutions to directly remove helices [13] would change composition guidelines in unpredictable ways. Polyvector fields with curl reduction [51, 145] can minimize, but fail to completely avoid, helices. Directly partitioning the mesh into quad layouts [23] could avoid helices but at the expense of a manual strategy that does not map well to how knitters make patterns.

Different quad and quad-dominant meshing techniques have been proposed for knitting. For example, [203] use a field-guided method to generate a stitch-level quad-dominant mesh but cannot ensure knittability because cycles cannot be fully avoided. [136] ensure knittability using a harmonic scalar field meshing technique that takes as input level set constraints of a scalar function approximating knitting rows. However, this method works at the stitch level and does not allow for composition or surface layout control.

In this work, we propose a new strategy to enable design control while avoiding helices in a coarse quad mesh. This is achieved by utilizing a key insight on the relationship between common knitting compositions and quad mesh singularities. In our system, composition guidelines selected by users are directly translated to singularity constraints on the mesh, which in turn can be used to drive a cross-field design algorithm based on trivial connections [41]. We choose to use pure quad meshes rather than quad-dominant meshes in order to exploit this singularity structure. Achieving knittability with quad-dominant meshes would require extra constraints aligning singular faces. We defer to Section 4.1.5 to introduce the details of our method.

4.1.4 System Overview

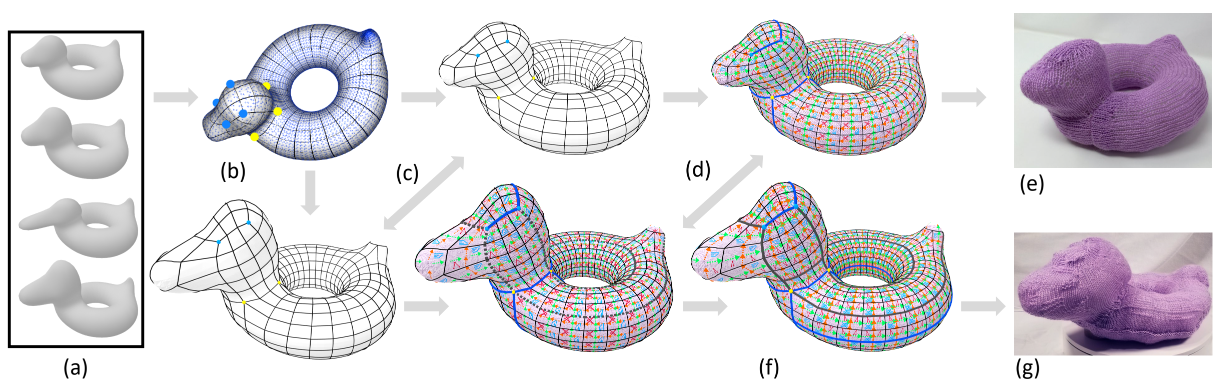


Figure 4.3: Overview of our framework. (a) Triangle meshes from a parametric template (the system deals with a single mesh at a time). (b) Input triangle mesh with user annotations of composition, layout, and direction guidelines. (c) Generated quad mesh patches which are consistent across template variations. (d) Quad mesh annotated for knitting the body tube in the round using short rows to curve the tube. Blue lines indicate seams. The same design applies to all template variations (two shown here). (e) Duck knit with short rows. (f) Quad mesh annotated with different textures and orientations; the body is knit as seamed sheets with decreases. (g) Duck knit with textures and a large head from template (f).

Figure 4.3 illustrates our system with an example of designing a toy duck. The system takes as input one variation from a parametric template of a duck mesh (**a**); if the geometry is not already a triangle mesh, the system will tessellate it into one. A parametric template is defined by parameters that describe degrees of freedom $\mathbf{q} \in \mathcal{A}$, where $\mathcal{A}$ defines the ranges of parameter variations that map to continuous geometric deformations. For our examples, we created the input parametric templates semi-manually by first creating cuboid cages in Blender and then computing the coefficients for interpolation [161]. Parametric templates can also be created with a variety of geometric editing methods [90, 69] or parametric computer-aided-design (CAD) tools. Our system allows users to create knit templates by enabling consistent control of design axes across the space of geometric variations.

The user starts by directly annotating knitting *composition*, *layout*, and *orientation* guidelines for how to break the duck into patches by indicating (blue dots in (**b**)) that the top of the head should be knit as a sheet to create a flap, that the head should be a tube abutting the body (yellow dots in (**b**)), and that the layout should be symmetric across the body. These composition guidelines are selected from an illustrated menu (Figure 4.5) and placed by clicking a position on the input mesh. They can also draw desired stitch orientation directly on the mesh, as well as explicitly specify layout boundaries as feature lines (not needed for this example). Our system maps these knitting directives to orientation, edge, and singularity constraints to create a novel coarse quad re-meshing algorithm that jointly re-meshes the entire parameter space of the input template to create a single parametric patch layout that satisfies the guidelines for all parameter values, (two shown, (**c**)). Users can choose to enable any available symmetries for their annotations, which encourages, but does not guarantee that the resulting quad layout generation will be symmetric.

This patch layout becomes the canvas on which the user designs their template. By clicking and dragging, they specify per-quad orientations, surface textures, and curvature shaping guidelines, as well as specifying seams. Users may also enforce exact symmetries by placing "equal stitch count" constraints on specific patch edges, which will be taken into account during the stitch generation step. Our interface is backed by a patch-level knittability solver built on top of Z3 [46] that not only validates the user's design but also assists in the design process by automatically finishing partial designs with knittable completions. For example, it can suggest seaming options based on existing annotations (Figure 4.4). In (**d**) the user has specified seaming-off the neck and knitting the inner tube with short rows, as well as a simple stockinette texture, while in (**f**) the user has rotated the orientation on the body, and our solver has assisted by finding an alternative seaming strategy that works with that orientation. At any point during design, the user can vary the template parameters to preview different customizations of their design. Finally, the user selects two variations to generate machine instructions for and fabricate (**e** and **g**). We convert the user-created Coarse Knit Graph to a Knit Graph and follow the procedure developed by Narayanan et al. [136, 137] to trace the graph and convert to machine instructions.

We will focus on the surface patch generation which utilizes the knit object representation motivated by the formative study results with knitters.

4.1.5 Interactive Surface Patch Specification

After loading a parametric template of a triangle mesh, the user seeks to automatically generate a coarse quad mesh that adheres to the composition, orientation, and surface layout guidelines. Our key insight here is that

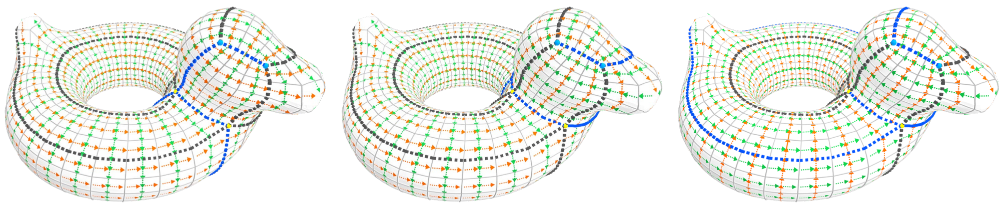


Figure 4.4: Interactive seaming suggestions. (left) Initial suggestion minimizing seams (blue lines), assuming no composition or layouts are given. (middle) Suggestion after the user specified that the full head would be a separate piece seamed at the neck. (right) Suggestion after the user changed knitting direction on the body.

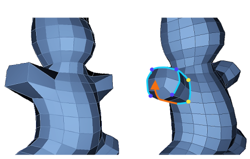
there exists a direct relationship between the singularity structure of cross fields and knitting composition guidelines. By identifying this correspondence, we created a quad meshing algorithm that is both theoretically grounded and able to represent important and commonly used knit pattern design techniques. Importantly, the method ensures the resulting mesh is helix free, a fabrication requirement of knitting, by providing feedback on knitting composition requirements, as well as a helix visualization tool to help users tune the grid size parameter towards a helix-free design.

Singularities and Composition Guidelines. The key composition guidelines used in knitting to assemble tubes and sheets map directly to singularities in the quad mesh. This is not surprising given their mutual correlation with the shape topology. We identified nine knitting-relevant composition rules (see Figure 4.5), each defining a set of singularities and seams whose indices either add up to -1 or +1. *Expansion rules* (-1 singularity sum) compose tubes together (E1—E4) or create a hole (E5—E6). *Contraction rules* (+1 singularity sum) close tubes with a slit seam (C1) or flap (C2), or bound sheets (C3). Contracting to a point is omitted as a +1 point singularity is not possible on a quad mesh.

This approach has several advantages.

First, composition rules describe common knitting patterns; for example, E1 splits a glove into fingers, and E2 is common in the armpits of sweaters. Therefore, knitters can work in terms they already understand rather than in singularities—they select composition rules from a menu and then click on the mesh to specify where they should be placed (see pilot study in Section 4.1.6).

Second, because which composition rules to use is associated with the template's topology, we can validate a composition and provide feedback on whether more expansion or contraction rules are needed: the sum of all singularity indices must be equal to the Euler characteristic, χ: $\sum_{v \in \mathcal{V}} index(v) = \chi = 2 - 2g - b$, where $\mathcal{V}$ is the set of vertices, g the genus number, and b the number of boundary loops. Importantly, giving control over composition allows the same shape to have multiple valid compositions.



For example, in the inset figure, the arm of the teddy bear model could be created by adding curvature to the body tube (left), which is more likely to fail on a machine, or knit by doing a merge and then a flap at the hand (right), which is a more natural design for knitters to come up with. Our interface allows users to have such high-level control while ensuring that the total sum of subscribed singularities is valid.

Finally, with these composition rules, no additional seams are necessary except on surfaces of non-zero

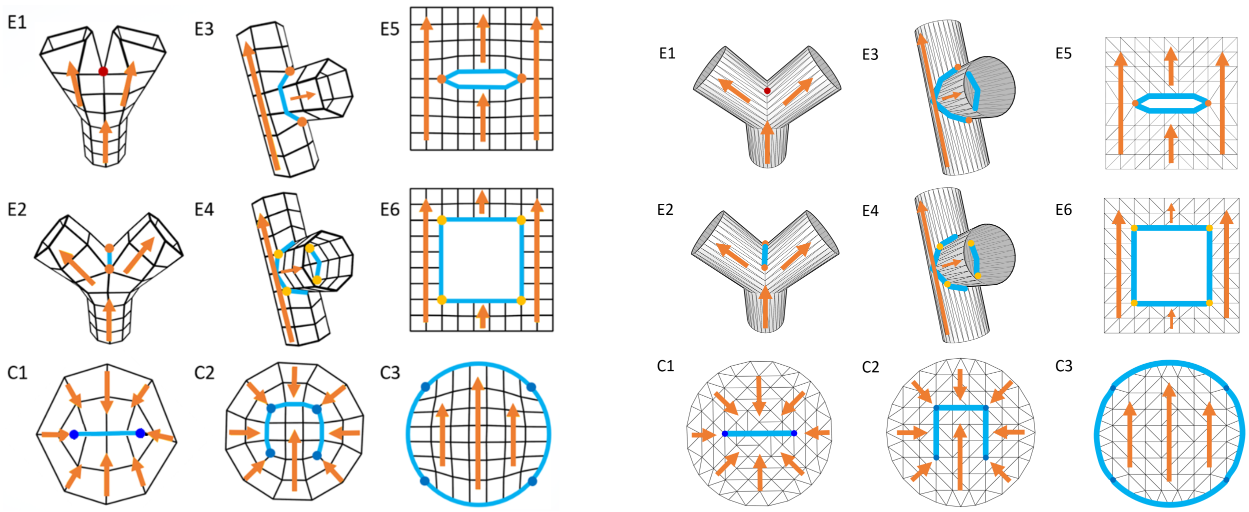


Figure 4.5: Knitting composition rules shown as quad singularities on remeshed quad meshes. The corresponding triangle mesh versions are shown on the right-hand side. Knitting directions are in orange and seams in blue. Expansion rules correspond to negative singularities (top two rows): joining tubes with no change of knitting direction can be done without seams with one -1 singularity (E1) or with a seam connecting two $-\frac{1}{2}$ singularities (E2); joining tubes with direction change requires closed seams, which can be done with two $-\frac{1}{2}$ singularities (E3) or four $-\frac{1}{4}$ singularities (E4); adding a hole without changing the knitting direction can be done with two $-\frac{1}{2}$ or four $-\frac{1}{4}$ singularities (E5,E6). Contraction rules correspond to positive singularities (bottom row): knitting in the round and joining at a line seam corresponds to two $+\frac{1}{2}$ singularities (C1); knitting in the round and closing with a flap that is knitted as a sheet and seamed along its boundary corresponds to four $+\frac{1}{4}$ singularities (C2); knitting a flat patch corresponds to four $+\frac{1}{4}$ singularities at the boundary (C3).

genus (e.g., a torus would need a seam to separate the first and last row), an additional benefit of our approach.

Controlled Meshing for Knitting Based on the correspondence analyzed above, composition guidelines selected by users define singularity constraints on the mesh. Our system uses these constraints to drive the trivial connections cross-field design algorithm [41].

Designers can further provide knitting direction guidelines by drawing directly on the mesh. *Soft directional guidelines* respect the existing composition and are treated as constraints on the trivial connections solver. *Hard direction guidelines* override the singularity structure imposed by the composition; the field is completely determined through cross-field interpolations, with these direction guidelines as constraints [155]. Hard directional guidelines are typically not necessary and often ill-advised because they may cause arbitrary singularities and create cyclic dependencies (helices). To give designers full control of the directional field, our method includes this option and checks for helices [14] providing feedback to designers.

We further let designers sketch directly on the mesh to place feature lines for surface layout control. We also allow easy specification of smooth closed loops using the method proposed in [23]. If feature lines are specified as seams, the mesh gets cut along them; this affects field optimization since there can be no smoothness constraints across seams. Otherwise, feature lines are treated as hard integer constraints in the integer grid optimization, which enforces placement of edges on the generated quad mesh.

Finally, these fields and constraints are used to create a mesh using mixed-integer quadrangulation (MIQ) [16]. The key modification that we make to the MIQ optimization relates to templates, which we discuss below.

Parametric Template Variations Parametric templates have been extensively used in the fabrication community to allow shape variability and customization while preserving manufacturability [162, 169]. Commercial systems also use templates for personalization, e.g., for 3D printing (`https://www.thingiverse.com/`). To generate a parametric knit template, we must define a consistent quad-mesh across the parameter space defined by template parameters $q \in \mathcal{A}$. By consistency, we mean that the user should define design axes only once, and they should propagate consistently throughout the full parameter space.

We assume that the user inputs a parametric template triangle mesh with point-wise correspondence—i.e., there is a bijective homeomorphism between $\mathcal{M}_q$ and $\mathcal{M}'_q$ for all $q, q' \in \mathcal{A}$, where $\mathcal{M}_q$ is a mesh representing the variation defined by q. This correspondence is directly specified when a parametric model is created by geometric deformations, and there are methods for constructing these maps for parametric CAD models [163]. Given a point-wise correspondence, a naive solution could define a quad-mesh for one shape and propagate the result. However, this may create quads with high distortion if variations are large. Consistent quad meshing has only recently started to be studied. Azencot et al. [7] proposes a method for consistent cross fields between two shapes with point-wise correspondence. However, this work would not allow us to preserve composition guidelines across variations since singularities and combinatorics of the final meshes may vary.

Our key insight of representing the composition axis as singularities makes consistent template generation possible by propagating the singularities with the point-wise correspondence and using them to drive the cross-field optimization on each mesh $\mathcal{M}_q$. We can then jointly solve for a parameterization using a variation on MIQ. As described by [16], MIQ takes as input a cross field, defined by two orthogonal vector fields $(\mathbf{u}_T, \mathbf{v}_T)$; it finds a parameterization onto an integer grid (u, v) by minimizing $\|h\nabla u - \mathbf{u}_T\| + \|h\nabla v - \mathbf{v}_T\|$ integrated over the surface, for some size parameter h and additional integer constraints derived from singularities. Since singularities are preserved in our method across template variations, we can use any value of q to define the integer constraints and minimize an energy summed over all variation of the mesh $q \in \mathcal{A}$:

$$E = \int_{\mathcal{A}} \int_{M} \|h\nabla u - \mathbf{u}_T^q\| + \|h\nabla v - \mathbf{v}_T^q\| dA dq$$

To solve this numerically, we discretize the inner integral as a sum over triangles and the outer one by sampling values on $\mathcal{A}$. Since computation could grow significantly with the number of samples, we solve MIQ in parallel across n different configurations of the mesh and add in a linear equality constraint that the (u^q, v^q) coordinate values should be equal according to the point-wise correspondence between meshes. Because these are linear constraints, we can use them to eliminate variables that are part of the MIQ solver. This makes the system matrix for solving n samples about the same size as for one sample.

4.1.6 Results

We demonstrate the effectiveness of our design system by a series of examples highlighting the capabilities in quality parametric meshing, creating a wide variety of objects, and interactively exploring the space of design variations for knitting.

Quad Meshing Other work has taken a field-based meshing approach to knitting, but ours is the first to explicitly incorporate singularity structure to control composition. The teddy bear example demonstrates the benefits of this approach. It would be natural to knit the teddy bear using tubes for each limb and one for the body and head. Achieving this composition from only user-provided direction strokes is very difficult because specific compositions need specific singularity placements, and singularities are difficult to control with only directional strokes. In Figure 4.6, the left two images are typical examples of a purely orientation-based meshing of the model. The inability to precisely control field singularities leads to helices wrapping around the body. These require long and unnatural seams to break the dependency cycles they induce. On the right is a structure resulting from the application of our composition rules, and the resulting knit bear.

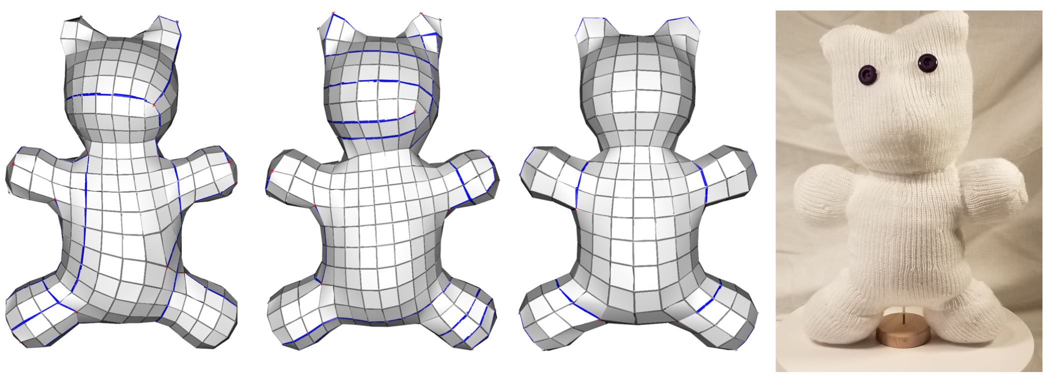


Figure 4.6: (left two) Meshing results achieved by sketching directions on the surface. Both have helices that must be seamed off and would be complicated and non-intuitive to sew. (third) The meshed teddy that our system generated to match the composition rules of knitting each limb in the round and then sewing them onto the torso followed by (fourth) an image of its physical realization as a multi-part knit. The blue lines are seaming suggestions proposed by our algorithm.

We also validate our joint parametric MIQ by comparing it to the naïve strategy of solving against a single mesh variation and propagating via pointwise correspondence. In Figure 4.7, all dresses have the same singularity structure. The dresses on the left were jointly parameterized using our approach, while the pairs on the right were computed on one dress and transferred to the second. Compared to the joint parameterization, transferring the child's pattern to the adult dress leads to distortions in the midsection, whereas the other direction has distortions in the bust and asymmetries in the skirt.

Design Space Coverage Next, we analyze our tool's coverage of the design space, based on the design axes that we have identified.

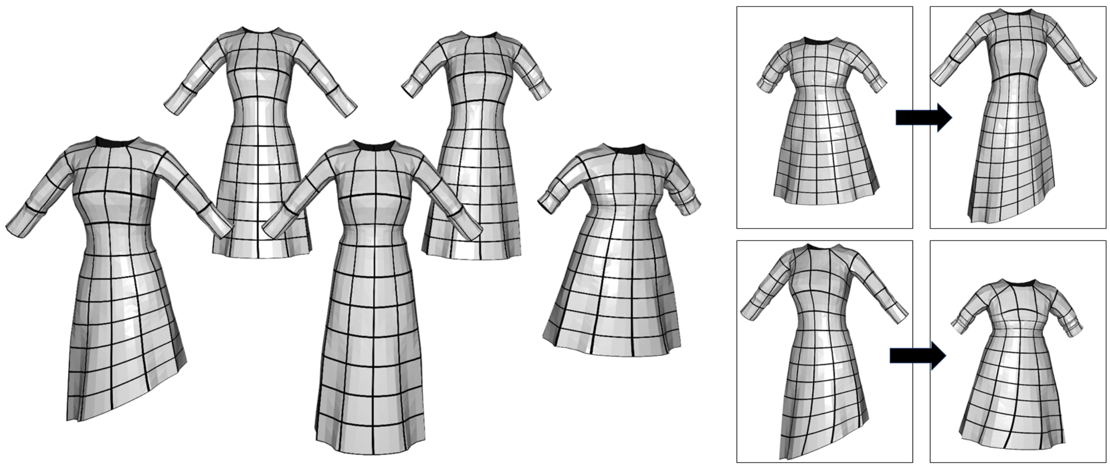


Figure 4.7: Our consistent meshing is shown on the left and compared to the naive approach of running the MIQ on one mesh and transferring the resulting quadrangulation onto another using the point-wise correspondences (two examples shown on the right).

Surface Texture Textures are illustrated in Figure 4.1 and Figure 4.8. As can be seen in both, texture does not need to be uniformly applied over the entire model but instead can be applied to any region that aligns with patch borders. This provides full control over texture since surface layout features can be used to influence border placement during meshing. Interaction of texture with other design axes can be complex. For example, textures and shaping can co-exist in the same patch and are automatically handled by our pattern generation algorithm. In Figure 4.8(c) the diamonds at the bottom are much wider than the diamonds at the top, due to decreases necessary to change the radius of the skirt from bottom to waist.

Curvature Shaping Shaping plays an aesthetic as well as a functional role. Figure 4.8 illustrates user control over how decreases are placed for different aesthetic results. Skirt (a) is fully symmetric, while skirts (b) and (c) have flat backs, with shaping only allowed on the front, and skirt (d) only has shaping at the sides with a flat front and back. Skirts (b-d) have decreases aligned and leaning toward a patch edge, creating an apparent seam.

Surface Layout As already described, surface layout features impact the placement and transitions between textures. Figure 4.8 shows another fundamental, though more subtle, impact on curvature shaping—the user prescribed feature lines to control the placement of apparent seams, as can be seen when comparing skirts (b) and (c). Symmetry is another important layout feature. Once this is specified, the system will automatically ensure symmetry in the placement of increases and decreases, seams, and even singularities on compositing guidelines. For example, in Figure 4.3 it would be hard to place singularities symmetrically without automated support.

Seaming Key functional aspects of seaming are specified during composition, but these interact through decisions about the surface layout of the mesh; seaming is often necessary to support orientation changes. Our system helps the user navigate this space. As was discussed in Figure 4.4, even if seaming guidelines

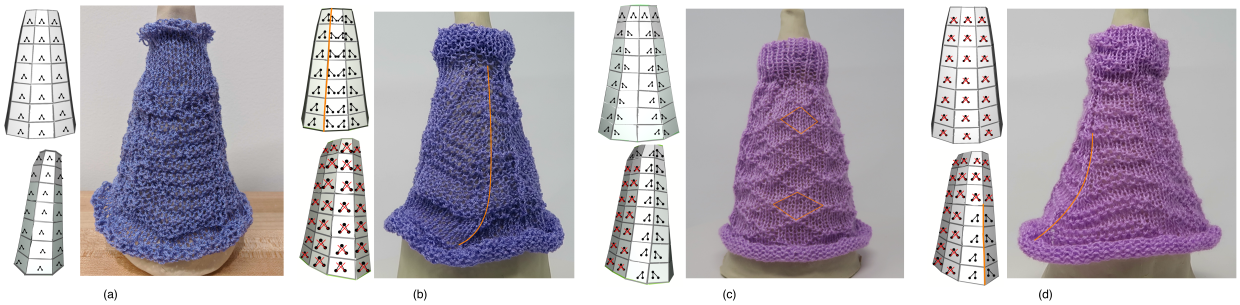


Figure 4.8: Meshing, labeling, and fabricated results of four skirts generated from the same input mesh, with different coarse meshing and labeling. Orange lines on (b) and (d) highlight the apparent seams caused by concentrating shaping, and on (c) highlight the effect of shaping to narrow the texture towards the top of the skirt. Red "X"s over a shaping label indicate that no shaping is allowed in that patch, a symmetric decal indicates distributed decreases, and an angled decal indicates leaning decreases aligned to the edge the decal leans towards.

from composition and surface layout are not provided, a valid and minimal seam choice will be presented to the user, who can interactively control the placement by enforcing or disallowing seams on certain areas. The system updates the seam suggestions at interactive rates, to allow easy exploration while guaranteeing knittability. Figure 4.9 show the seaming layouts of our machine knit examples with seams.

Orientation As discussed in multiple examples, orientation affects the ease of knitting, shaping choices, and seaming placement. Further, local changes in orientation can lead to non-fabricable designs if not validated globally (an important reason orientation change isn't supported in [137]). In addition to allowing orientation control during meshing, our system allows users to easily flip the orientations locally. As can be seen in Figures 4.3 and 4.10, the system will automatically suggest seams after a direction change to ensure knittability and update shaping to conform to the mesh—e.g., use short rows instead of increases/decreases on the sleeves to match the circumference change from shoulder to wrist.

Composition Composition guidelines allow designers to create large pattern variation from the same input mesh. For example, in Figure 4.11, the design on the left uses a Norwegian drop shoulder and is seamed at the arms. The design on the right is a seamless yoke sweater, which is done with merges at the armpits followed by evenly distributed decreases up to the neckline. A similar composition variation is shown on the dresses (a) and (b-d) in Figure 4.1. Composition guidelines are particularly useful when knitting complex shapes, as discussed in Figure 4.6. We further illustrate how they can be used to structure irregular shapes like the bunny (see Figure 4.12). By specifying how we wish to knit the ears and tail, our system discovers appropriate knitting directions to capture both the compositional structure and the complex curvature.

Variable Shape. As discussed in Section 4.1.2, resizing is an important and common aspect of knit pattern design and use. Resizing is challenging because it requires changing both the stitch counts of shaping operations, such as the number of short rows, increases, and decreases, as well as re-applying any textures to the new stitch layout. Further, resizing typically requires variations on the geometry itself. A dress made to fit a child is not simply a rescaled adult dress.

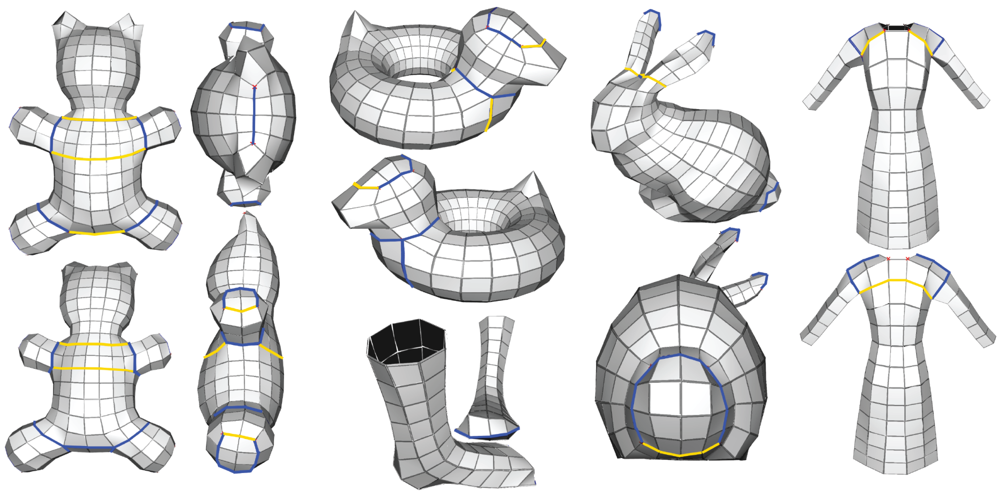


Figure 4.9: Seaming layouts of all machine knit examples. Blue lines indicate seams in the original design, whereas yellow seams were added manually to account for missing functionality in our scheduler implementation. The skirts, hats, and seamless dress were omitted because they do not have any seams. The angled and child dress are ommitted because they use the same template as the drop shoulder dress (far right). The human scale sweater also uses this seaming layout.

Our method allows users to create a mesh that is jointly optimized over multiple parameter values of a shape, which allows users to specify knitting guidelines on a single template and have them be directly applied to different shape variations. For example in Figure 4.1, the adult and child dresses are variations of the same drop shoulder pattern with identical textures but different relationships between arm length, skirt length, and torso height. Figure 4.13 further illustrates how our system allows designers to create customizable templates for knitting, by illustrating three fabricated variations of a hat. Both of these examples were enabled by our consistent quad-meshing method, shown in Figure 4.7.

Interactive Exploration All models took about 4-13 minutes to design: the quad meshing step took about 1-6 minutes, the labeling about 1-3, and pattern generation with sizing optimization in 2-4 minutes (except the bunny, which took 15 minutes to optimize sizing). To establish the effectiveness of our interactive editing capabilities for design space iteration, we asked Narayanan to recreate some variations of the dresses in Figure 4.1 using [137]. In our system, we were able to create an initial design in 5 minutes, and create the variants (c) and (d) in 2 minutes each, most of which is spent in pattern generation. Variant (a) took 8 minutes as it required composition changes. Narayanan estimated that it would take between 15 and 40 minutes for each texture variation, depending on how carefully textures were applied, and between 45 and 60 minutes to change shaping between short rows and increases and decreases. Their system would not be able to handle direction changes or re-sizing without complete re-design. This shows how our approach and solver assisted editing enables exploration of design alternatives on the scale of minutes rather than hours.

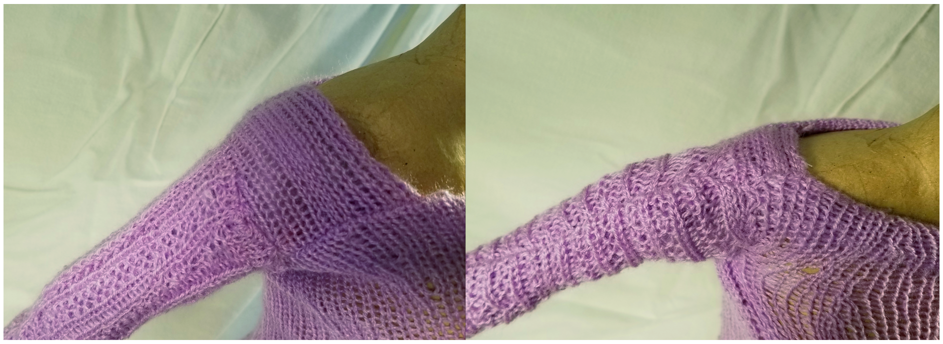


Figure 4.10: Close up of the sleeves from Figure 4.1 dresses (b) and (c). In (b), columns are aligned down the sleeve, while in (c) they wrap around.

Figure 4.11: Three sweater patterns from the same input model but meshed and labeled differently. The left sweater has seamed sleeves and short rows on the neck, while the one in the middle was completely knitted in the round using increases and decreases. These two sweaters were knitted by hand, showing how our method can be used for both machine- and hand-knitting. The rightmost sweater has the same composition as the leftmost, machine knit to human scale with textures added on the arms.

Pilot Study We validated the usability of our system by conducting a pilot user study with three participants having experience in knitting or garment design but not in geometry processing. In the study, we first gave a tutorial on how to use our system and then asked the participants to reproduce a textured variant the duck design shown in Figure 4.3 (e), and also to create their own dress design using the model from Figure 4.1. All participants were able to determine the correct composition rules to recreate the duck within 9 minutes on average, and were also able to design a knittable dress within the half-hour provided them. As shown in Figure 4.14, all three dress designs have a different composition structure. While the users had no understanding of quad-meshing singularities they managed to achieve the desired structure using the intuitive composition guidelines in our tool. These dresses further illustrate the design freedom in textures, shaping, seaming, and surface layout.

Additional Implementation Details With the user-designed cross field as input, we use the libigl [91] implementation of MIQ to generate the global parameterization; users can adjust a parameter for quad size to

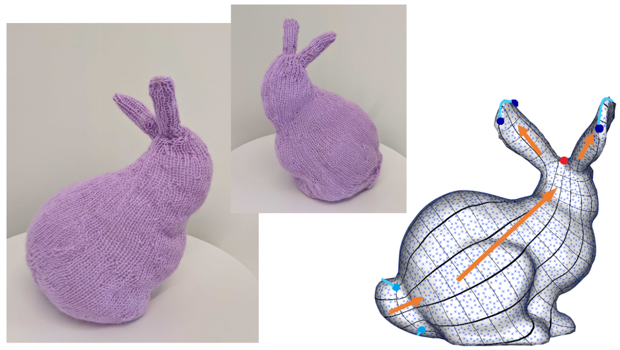


Figure 4.12: The Stanford bunny illustrates composition guidelines. It is meshed by placing line seams (C1) on the tips of the ears, a split from the head to two ears (E1), and a flap on the tail (C2).

Figure 4.13: Customizable knit patterns for hats created with our systems.

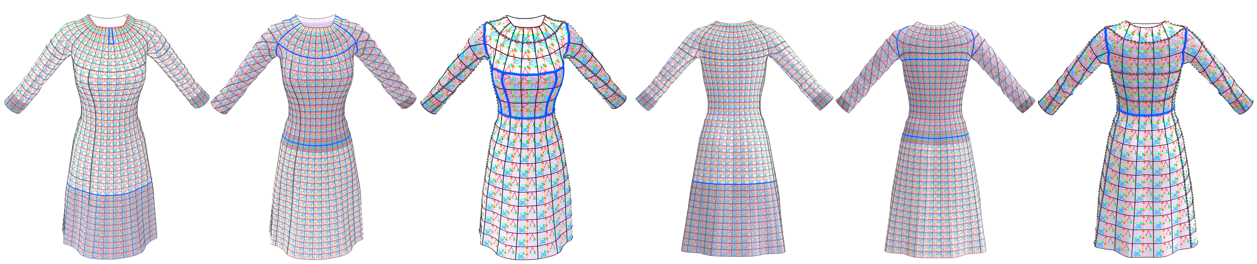


Figure 4.14: Variations of the dress created by users in the pilot study (left three is front view, right three back view), illustrating the usability and design freedom in the system. Users were able to control composition using only prior experience on knitting or garment design but no understanding of geometry processing, quad meshing or singularities.

obtain the desired level of coarseness. To compute the cross-field given composition rule singularities, we use the implementation of the trivial connections from Directional [192]. LibQEx [54] is used to extract the quad mesh. Z3 solver [46] is used for SMT equations. We implemented 20 textures from [105] and applied different combinations to most of the models we fabricated to illustrate this capability. For machine knitting, we use the scheduler code provided by Narayanan et al. [136, 137] to generate instructions for the knitting machine. Hand knitting instructions were generated using custom code. All examples are knitted either by hand or by a 7-gauge SHIMA SEIKI SWG091N2 knitting machine. Models were hand-stitched together along seam lines after knitting, and the toy models were stuffed with batting.

4.1.7 Discussions and Conclusion

Our system makes the design of machine- and hand-knittable objects accessible to a lot more people. First, it lets users easily and quickly explore interrelated design axes while guaranteeing knittability and pattern production. Second, because the system takes a parametric 3D model as input, it generates template patterns customizable by users unfamiliar with intricacies of knitting. As a result, machine knitting, like 3D models, can become customizable, modifiable, and universally accessible, enabling more designs that can be fabricated through knitting – a material-efficient fabrication method. Although quantifying the material usage was not the focus of this work, future work could consider measuring the material cost fabricating a garment pattern designed using our method. This metric could then be used to inform the parametric template design and potentially also stitch-level optimization.

4.2 Case Study: Design with Repurposed Everyday Objects

4.2.1 Introduction

In nature nothing is lost, nothing is created, everything is transformed. —*Antoine Laurent de Lavoisier*

Everyday life presents many challenges regarding our physical environment that we are constantly trying to solve, from common wear and tear (such as stovetop stains) to cluttered spaces (such as a messy desk). It is tempting to purchase the latest cleaning or organizational tools in a world of next-day delivery that bombards us with advertisements. However, buying still more products is wasteful, costly, unsustainable, and often unnecessary. Instead, a thriving subculture is growing on the Internet of sharing "home hacks" that repurpose common household items into cost-effective and environment-friendly solutions. This "fabrication process" consumes no new materials and is thus a sustainable process. In order to build a design system that supports such a process, we need to come up with a representation for home hack designs.

We analyzed the space of home hacks (full analysis in Appendix A.2) and found that we can divide them into two categories based on their functionalities. One category, like a blinds-cleaning tool made from binding tissues on tongs with rubber bands, makes creative reuse of a *single item* to change the shape or feel of an existing object, enabling better grasping or easier interaction. The other, like hangers linked with soda

Figure 4.15: We created FabHacks, a design system for "home hacks" built from repurposed everyday objects. The system is built on FabHaL, our domain-specific language for representing rigid fixture hacks. This solver-aided DSL is equipped with verification and solving functionality to help the user finalize their designs. Here we show two hacks, each with the set of everyday items to build it, the solved configuration from our system, and the design fabricated in the real world. Left: the birdfeeder hanging hack made of S-hooks, eyehooks, sticky hooks and a hanger. Right: the reading nook hack made of obstacle rings, toy ring links, S-hooks, turnbuckles and a hula hoop; the environment for the reading nook hack was scanned and calibrated with the PolyCam mobile application.

can tabs to make effective use of closet space, involves *assembling multiple items into a structure* that holds objects at a specific location and orientation relative to the environment. We term the latter "fixture hacks" because their goal is to build an assembly that holds a *target object* in a fixed *environment*. Our analysis found that rigid undeformed fixtures (i.e., composed of rigid parts combined but not deformed or modified destructively) are typically used in fixture hacks. Thus, our work focuses on this well-scoped subset.

Replicating existing fixture hacks at home might be straightforward, but inventing new hacks requires knowledge, insight, creativity, experimentation, and access to all parts. Furthermore, fixture hacks often involve multiple objects that interact mechanically, and gravity can affect a design's stability, making physical prototyping necessary to design a hack. However, physical prototyping is not always possible, not only for people with limited mobility, but also in situations where not all parts are available or prototyping would be costly or permanently alter one's home.

Our main insight is that despite the variety of objects used in rigid fixture hacks, these objects attach via eight common types of *connector primitives* (Figure 4.16). For example, the handle on a mug, the top hook on a hanger, and the handle on a basket can all be represented using a "hook" primitive (Figure 4.18). The connector primitive is thus a key concept in our system: these primitives abstract away the complex low-level geometry that is irrelevant to how users combine objects or to the overall assembly functionality.

This insight informed the design of *FabHaL* (FabHacks Language), the key contribution of our work. FabHaL is a solver-aided domain-specific language (S-DSL) for representing fixture hacks. By embedding the

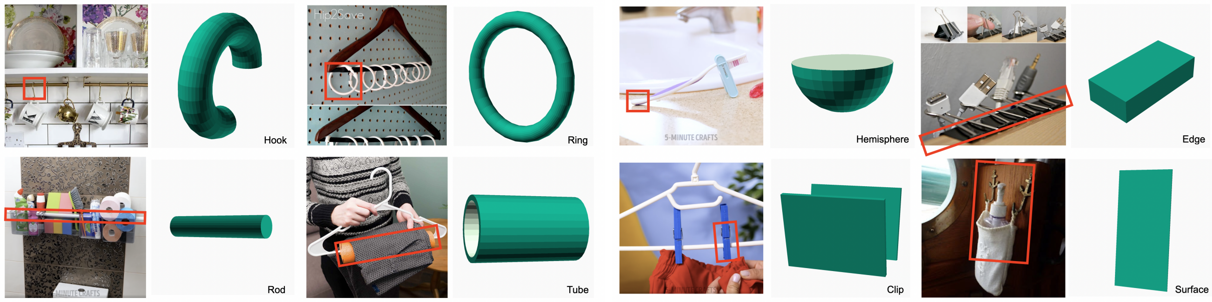


Figure 4.16: We analyzed 24 rigid undeformed fixture hacks and extracted eight connector primitive types found on objects in those hacks; Table A.4 in Appendix A.2 documents the shape parameters we use to parametrize a primitive's geometry. We show each connector next to an example hack where it appears. The eight example hacks (left to right, top to bottom) are cup hanger (No.24), scarf organizer (No.22), toothbrush holder (No.7), charger holder (No.18), bathroom organizer (No.11), nonslip hanger (No.3), pants hanger (No.8), and soap bottle bag (No.1) as numbered in Table A.2 in Appendix A.2.

connection behavior and compatibility constraints for each pair of connectors into the solver (Section 4.2.4), we help users more easily explore hack designs within the domain's constraints.

On top of FabHaL, we build a novel design system and UI, *FabHacks*, for designing home hacks. The system lets users experiment virtually and simulate their designs under gravity. We validate our system through a user study, with results showing that users find FabHacks intuitive to use and is useful for exploring hack designs.

4.2.2 Related Work

This work proposes FabHacks based on the FabHaL DSL that addresses the specific challenges posed by the domain of fixture hack design. We survey tools and recent work related to our approach.

CAD Tools for Assembly Design Assembly design is important in manufacturing industries. Various tools have been developed for this task, including computer-aided design tools [142, 172]. We can use CAD tools to construct assemblies of parts using *mate constraints*, which define the relative orientation of two entities (part or surface) and the constraints on their degrees of freedom. However, modeling complex assemblies with existing CAD software requires a high degree of expertise.

First, mates are tricky to work with despite recent research [96] on providing mating suggestions. Multiple different mate types between two parts could appear to encode the same kinematics, only to be shown different later in the design process when another part is added that further constrains the existing degrees of freedom. Mate constraints are also not made for representing fixture home hacks. The everyday hacks that inspired our system (Figure A.2) consist of many loose connections, such as a hook dangling over a rod, or a ring with a much greater radius than the hook it is attached to. Mates, usually single-origin coordinate systems with limited degrees of freedom, are more suitable for representing a mechanical assembly where parts fit snugly together, leaving only a few degrees of freedom for the overall assembly motion.

Second, if the object geometry comes in other formats (such as point clouds from scans, voxels, or inaccurate STLs), CAD users must create B-rep models from these inputs before they can specify mate connectors. FabHacks can accept any format and simply requires the geometry to be tagged with connector primitives. For example, for the reading nook hack in Figure 4.15, right, we scanned the room and used it as the geometry for the environment. We created an OnShape plugin (Figure 4.18, top) for tagging the connector primitives on geometry that comes in various formats.

Finally, performance analysis is also important during assembly design. Existing CAD tools are primarily concerned with analyzing the kinematics of mechanical assemblies and evaluating whether they achieve the desired concerted motion. In contrast, evaluating the performance of rigid fixture hacks that we focus on means measuring their stability as a hanging assembly under gravity. This type of simulation-based analysis is either completely separate from current CAD design tools or exists with the CAD tool as part of a software suite that requires additional expertise to use.

Solver-Aided DSLs DSLs have proven effective at abstracting expert knowledge and allowing non-experts to create valid designs, but they are, by definition, designed for a specific domain of applications. Several works [219, 98] have used DSLs for geometric modeling in specific domains, like simulated terrestrial robots and cuboid-based 3D shapes; they define a DSL and try to synthesize programs in it given specific objectives. DSLs can also be used to specify designs and fabrication plans for carpentry [202, 220], where we can use program synthesis techniques for design generation and optimization.

In this work, we propose a DSL (FabHaL) specifically for fixture hacks. FabHaL imitates the paradigm of *solver-aided languages*, where a user can partially specify a program (vastly reducing the search space) while leaving certain sections abstract (such as expressions or parameters) [184]. An external solver is then invoked to concretize the partially specified program into a complete one, which can then be executed to verify the result. A FabHaL program is essentially a partial specification of a home hack design: a sequence of instructions to attach a specific connector primitive on one part to a specific connector primitive on another.

This paradigm has proven useful in domains in the programming languages community, such as program deobfuscation [95], synthesizing GPU kernels [148], and validating and planning biology experiments [66]. Other applications include user interface designs [84] for resolving conflicts in the constraints of a layout design and mathematical diagrams [212] for automatically placing visual elements given user specifications. In our case, users can specify the skeleton of connections between primitives while leaving the precise placements of parts for a solver to fill in, and the solver could also provide feedback to users, such as informing them of whether a connection is valid.

Generative Design of Connectors Existing works on modeling or creating connections involve generating new connection geometry. Koyama et al. [106] propose a tool for automatically generating 3D printed structures given a user specification to hold or connect two objects. Hofmann et al. [82] also generate connections between objects and support the specification of assembly information and constraints affecting the assembly, but they do not automate solving for those constraints. In addition, both works focus on manufacturing new parts, in contrast to our focus on exclusively reusing existing objects.

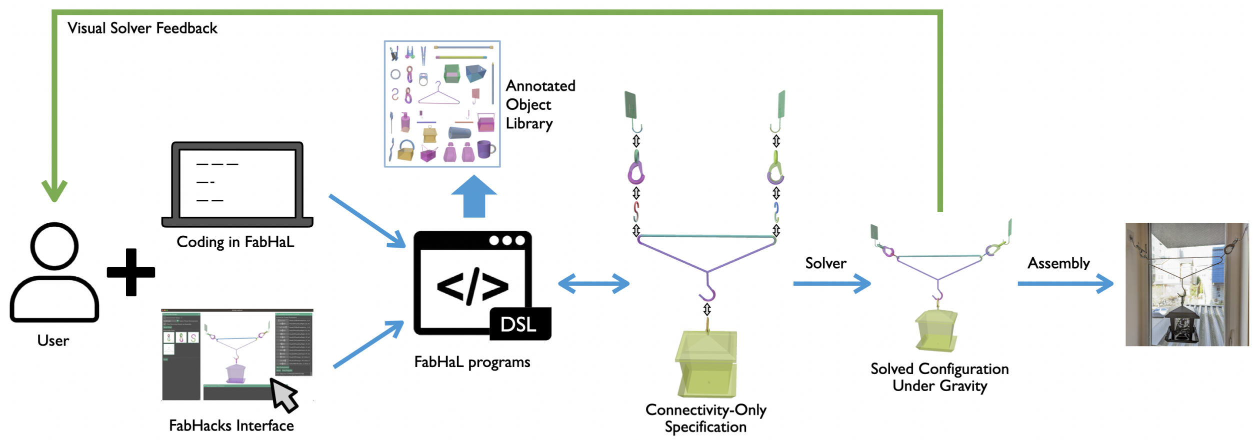


Figure 4.17: Overview of the FabHacks system. The user can either directly code in FabHaL or use the UI to create programs. FabHaL programs build on top of an annotated object library. The programs are connectivity-only specifications of a hack design, and the 3D configurations of the parts are completed by the automatic solver. Users can then get visual feedback from the program viewer and use the feedback to iterate on the design. When satisfied with the design, they can fabricate the hack in the real world.

Sustainability in Design and Fabrication Sustainability considerations have become increasingly prevalent in our everyday lives and in fabrication research [207]. Our work explores the general question of how to fabricate more sustainably. In this space, prior work explored how fabrication can reduce waste by using 3D printing to fix broken objects [183, 107] and reusing materials, such as plastic bags [35] and yarns [205]. Other work augments existing objects with fabrication for repurposing [45, 153, 73], such as by generating structures for re-interfacing with robot arms, legacy physical interfaces, or appliances. Chen et al. [33, 34, 32] use 3D printing to augment existing objects with additional functionality (some involving mechanisms), while Arabi et al. [4, 5] and Li et al. [115, 114, 116] focus more on augmenting robots using everyday objects or mechanisms to help them manipulate objects.

Our research examines how to use rigid everyday objects of any shape without modifications to build a hanging fixture. Our work is distinctive in that we consider how *multiple* objects fit together into an assembly; the preceding work instead augments one specific object to allow robotic manipulation or to create a mechanism. (For example, none of the preceding work could be used to design the hanging birdfeeder in Figure 4.15, which uses several different parts.)

4.2.3 System Overview

Consider as an example a novice user designing a birdfeeder to hang between two hooks using FabHacks (Figure 4.17).

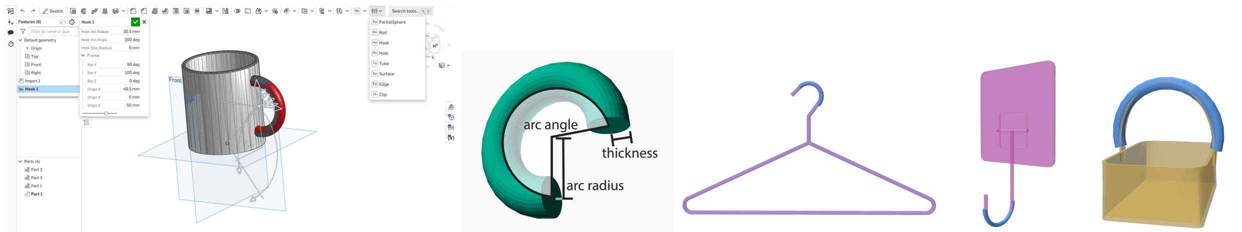
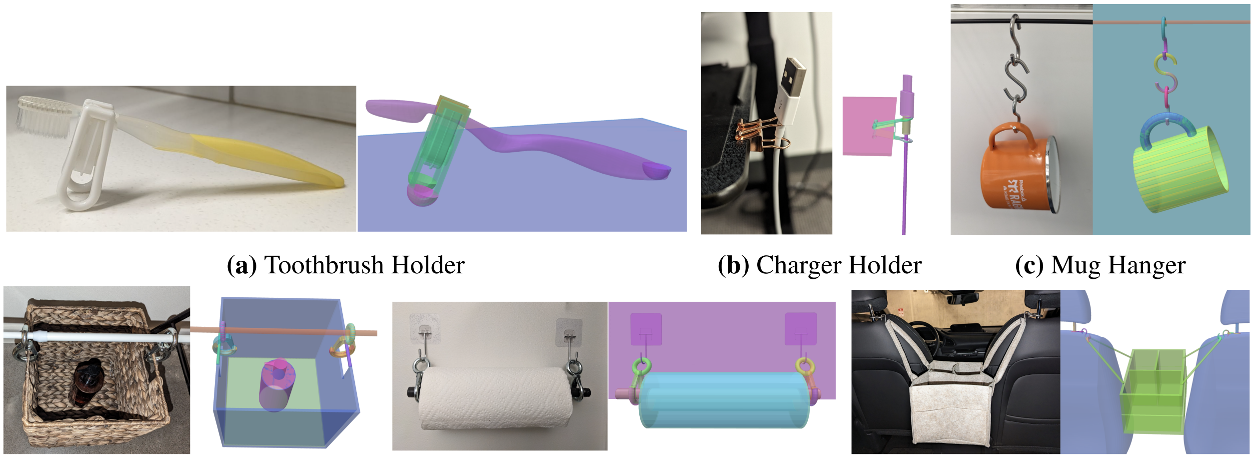


Figure 4.18: Top: the OnShape plugin for tagging 3D models with the eight connector primitives. Bottom: (left) an example showing how we defined the hook shape parametrically with its arc angle, arc radius, and thickness, and (right) three parts tagged with a hook primitive, each with different parameters.

Annotated Object Library First, the user selects the parts they would like to repurpose into their home hack from the *Annotated Object Library*. The Library contains 3D models of a variety of rigid everyday objects, each annotated with the eight types of connector primitives we currently support. We call these annotated Library objects "parts." In addition to labeling regions of a part with a connector primitive type (such as "hook"), the annotations include type-specific parameters needed to define the geometry of that primitive. For example, we show in Figure 4.18, bottom right, three example parts that have been annotated with a hook primitive, each parametrized to match the exact radius and thickness of the hook geometry in that part (bottom left).

We stress that the user does *not* typically need to do any 3D modeling or annotation themselves but rather can select parts from the predefined library. All examples in this work use a proof-of-concept library of 47 parts: 22 parts for to modeling the fixed environment or the target object to be held fixed in place by the hack, and 25 everyday objects rich in connector primitives for use as components of a home hack. To build this database, we extended the OnShape CAD modeling system's API to support part annotation. Our plugin (see Figure 4.18, top) lets users import a 3D model of a part and add connector primitives. When a primitive is added to a part, parameters are set interactively to ensure the connector aligns with the part. The Annotated Object Library can be extended to include custom parts that users want to include in their hack design, and we envision this to be part of a future community effort.

FabHaL Next, the user assembles parts into a hack design using FabHaL, our solver-aided DSL. Users have two ways to interact with FabHaL to create hack designs: either directly writing programs in the FabHaL language, or using the FabHacks graphical interface to click on two connectors of two parts to connect them. When programming in FabHaL, users can also *parameterize* the programs; for example, they can specify that a hack should include a chain with an unknown number N of links and search over N for valid hack designs with the help of the solver (Section 4.2.6). In either case, note that the user need not write any kinematic constraints: these are inferred automatically by FabHacks from the part annotations. See Section 4.2.4 for more information on the FabHaL language.



(a) Toothbrush Holder **(b)** Charger Holder **(c)** Mug Hanger

(d) Soap Bottle Holder **(e)** Paper Towel Holder **(f)** Diaper Caddy

Figure 4.19: Six hacks created by directly programming in FabHaL, with photos of fabricated designs and renderings of the corresponding programs in our viewer (see Appendix A.3).

Solver-aided Evaluation Finally, the user asks FabHacks to realize the hack design in 3D space using a constrained optimization solver (Section 4.2.4). Our solver checks whether the part connections are feasible and, if so, relaxes the 3D positions of the parts under gravity and presents the final, solved configuration visually to the user. The solver also reports problems with the design to the user (such as infeasible connections or parts that would fall off the assembly if relaxed under gravity). Given this feedback, the user can iteratively improve the FabHaL program and solve again.

4.2.4 An S-DSL for FabHacks

We now introduce the S-DSL FabHaL for representing rigid fixture hacks. Figure 4.19 shows six example designs represented in the DSL.

The design of FabHaL was motivated by two factors. First, our analysis of home hacks (see Appendix A.2) influenced the language design. We found that objects in home "fixture hacks" are typically connected via eight common shapes, which we term *connector primitives*, and define in Section 4.2.4.

Second, the language design is guided by our goal of using the DSL as a vessel for domain knowledge. We intend for this DSL to help users without prior experience in modeling or simulation to design fixture hacks. Therefore, being straightforward and succinct is an important desideratum. To achieve this, we choose to introduce a solver to complete a connectivity-only partial specification of the hack design, so the user need only specify (1) the configuration for a target object and its environment, and (2) which connector primitives to connect. We introduce the simple syntax and example usage in Section 4.2.4 and the solver in Section 4.2.4.

Connector Primitives

FabHaL includes eight primitive types: hook, rod, ring, tube, hemisphere, clip, edge, and surface, which can be assigned to a wide variety of objects (Section 4.2.3). We summarize the connectivity between these primitives in Table 4.2. Next, we explain how we model the connection behavior between pairs of connector primitives and the information associated with each primitive that the solver uses to verify and finalize the configuration of a hack design.

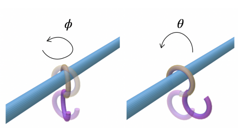
Table 4.2: Pairs of primitives that can be connected. A checkmark means that connection is currently allowed by the DSL, and a light grey cell means it is not. We ignore the lower-triangular region (dark grey) since it is redundant with the upper-triangular one.

	hook	ring	hemi.	edge	rod	tube	clip	surf.
hook	✓	✓			✓	✓		
ring					✓	✓		
hemi.								✓
edge							✓	
rod						✓	✓	
tube						✓	✓	
clip								
surf.								✓

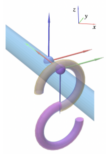
Figure 4.20: Three examples of rod-hook connections. Image on the right © Matt Kingston.

Connector Frames Our analysis of home hacks (Appendix A.2) found that the connection behavior between parts is local to the pair of primitives that form the connection. Take as an example the rod-hook connection (see inset): a hook can slide along a rod and flex around it regardless of whether this rod is in a closet, a shower, or an ironing board. To represent such behavior mathematically so that we can formulate it as part of the solver's constrained optimization, we must first establish the concept of a `Frame`.

In FabHaL, `Frame` consists of a position vector (x, y, z) and yaw-pitch-roll intrinsic Euler angles. Frames can represent a single-origin coordinate system (similar to mates in CAD) or the 3D configuration of a primitive or a part. When frames are used to represent the connection points on primitives, we call them *connector frames*. The connector frames of a primitive can be computed from its base frame and shape parameters (obtained from the part annotations) and additional degrees of freedom specific to its type. For example, a hook primitive has two additional DoFs, θ and ϕ, parameterizing the location and orientation of the point of contact (see inset). In FabHaL, the DoFs and the information on how to use them to compute the parametric connector frames are associated directly with each connector primitive.



Alignment Offsets When two primitives are connected, their connector frames need to be coincident in position, but the orientation may have some offset. Based on our analysis, this orientation offset is common to a



pair of connectable primitives. For example, as shown in the inset figure, when a rod and a hook connect, their connector frames are offset by a rotation of $[180°, 0°, 90°]$ in yaw-pitch-roll intrinsic Euler angles. (Here the frames are intentionally placed to be not coincident at their origins to better display the orientation offset.) We call the offset rotation between two primitives' connector frames an *alignment offset*.

With the connector frames and alignment offsets defined for each pair of connector primitives, we can represent the connection behavior precisely with respect to the degrees of freedom associated with each primitive. Even with a small number of categories of connector primitives, we can capture a wide range of possible connections that appear in hacks. This set of primitives and associated alignment offsets is also easily extensible.

In addition to the theoretically allowed connectivity between primitives (Table 4.2), two primitives must be physically compatible before they can be connected. We encode two pieces of additional information in connector primitives so that users need not reason about this lower-level detail.

Closed Primitives Two primitives with no openings cannot connect because there is no valid motion path to create the connection. Among the eight connector primitives, the ring primitive is always closed. In addition, primitives that are not generally closed could be inaccessible in the context of the geometry of the part containing it. For example, the handle of the basket in Figure 4.18 (bottom-right) is tagged as a hook, which can connect to a ring according to Table 4.2. But as an integral part of the basket, it is part of closed geometry; thus, a ring primitive without an opening cannot connect to this hook. We allow tagging of individual primitives as *closed primitives* (e.g., the basket handle) when annotating parts for the Annotated Object Library, and our solver checks that designs do not attempt to connect two closed primitives.

Critical Dimensions Primitives might not have sufficient physical space for a connection. For example, a one-to-one connection between a rod and a hook is possible only if the hook's hoop radius exceeds the rod's radius. For a multi-to-one connection between several hooks and tubes and a single rod, the hooks and tubes might fully occupy the length of the rod. In this case, no new connection could be made with the rod because there is insufficient space.

To track available physical space on primitives, we specify a *critical dimension* for connector primitives that can have multiple connections (i.e., the eight primitives except hemisphere and clip). The *available critical dimension* refers to the dimension of a primitive that gets occupied when a new connection is made between itself and another primitive. For example, the critical dimension of a rod is its length; when a hook connects to this rod, we reduce its available length by the hook's width. The hook's critical dimension—the hoop radius—is also reduced by the rod's radius.

Language Constructs and Hack Construction

To represent a hack, we must connect *parts* (annotated objects from the Annotated Object Library) using their connector *primitives*. These connected parts form a graph (Figure 4.21) that we call an *assembly* (i.e., a hack).

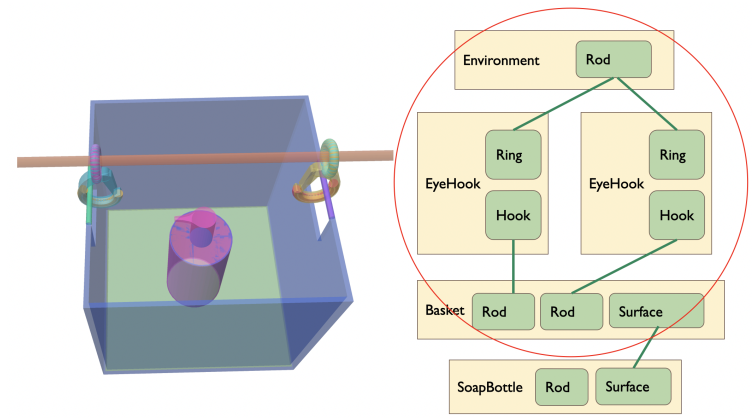


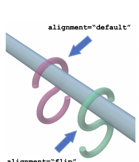
Figure 4.21: An assembly with a cycle: a basket is connected to a rod via two eyehooks, forming a cycle (in the red circle) between the basket and the environment. Yellow rectangles represent parts, and green ones represent primitives.

Two special parts in an `Assembly` are assumed to be fixed in place: the part representing the *environment* the assembly is attached to, and the *target part*, a part meant to be fixed relative to the environment and whose configuration is used as a target for the solver. For example, the clip in Figure 4.19a is rests on a table, supporting a toothbrush. The table is the environment, represented as a surface primitive with a fixed position and orientation. The toothbrush is the target part to be fixed above the table.

Our DSL exposes three operations needed to create an `Assembly`:

- `start_with(part, frame)`
- `end_with(part, frame)`
- `connect(part1.primitive, part2.primitive)`

`start_with` is used to specify the environment part with a fixed configuration (frame), and `end_with` specifies the target part's configuration (frame). `connect` takes two primitives as arguments regardless of order and determines whether each `Part` is already part of the `Assembly` or is newly introduced. It has two



optional parameters: (1) `alignment` (either "flip" or "default") to indicate an orientation flip, e.g., a hook can hang on a rod coming from both sides of the rod, as shown in the inset; (2) `is_fixed`, a boolean value that indicates that the connection is formed by static friction and thus the degrees of freedom involved should be held constant during solver-aided evaluation (e.g., the design requires taping connectors together).

If both connected parts are already part of the assembly, this connection creates a cycle in the graph representation of the connected parts (see Figure 4.21). Not all `connect` operations will be physically realizable, and we discuss how the solver verifies whether a connection can be made in Section 4.2.4.

```
assembly = Assembly()

# adding the starting environment
ENV_start = Environment({"part": TowelHangingEnv()})
start_frame = Frame([0,0,200], [0,0,-90])
assembly.start_with(ENV_start, start_frame)

# connecting the eyehooks to the starting environment
eyehook1 = HookEyeConnector()
eyehook2 = HookEyeConnector()
assembly.connect(eyehook1.hole, ENV_start.rod1, alignment="flip")
assembly.connect(eyehook2.hole, ENV_start.rod2)

# adding the target part and connecting the eyehooks to it
ENV_end = Environment({"part": RoundHandleBasket()})
assembly.connect(eyehook1.hook, ENV_end.hook)
assembly.connect(eyehook2.hook, ENV_end.hook)
assembly.end_with(ENV_end.hook, Frame([0,0,100], [-75,0,90]))
```

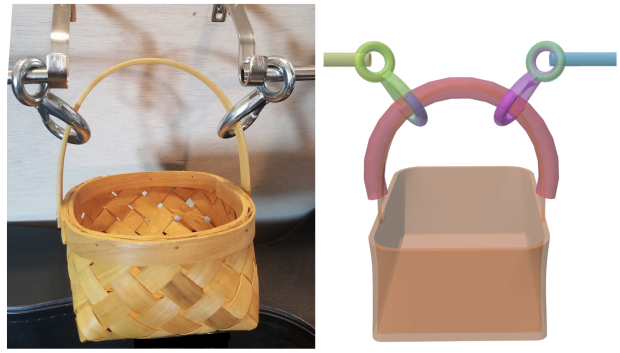


Figure 4.22: An example program in FabHaL (top), with the corresponding assembly solved for and physically reproduced (bottom left) and rendered by our system (bottom right).

Figure 4.22 shows an example program in our DSL. This fixture hack hangs a basket with a round handle between two rods. In this program, we first initialize an `Assembly` and then the environment. Next, we use `connect` to add two eyehooks to the two rods by connecting the eyehook's eye to the rod. Finally, we initialize the target part and connect the hook part of the eyehooks to the handle of the basket.

Solver-aided Evaluation

The core advantage of FabHaL is its ability to simplify the representation of an `Assembly` to a graph of connected `Parts`, leaving to the solver the work of calculating the placement of parts.

In our solver, we model an `Assembly` using a reduced representation of a kinematic rigid body chain, a common approach in fields like rigid body mechanics and robotics [65]. Except for the environment part configuration, which takes 6 DoFs (degrees of freedom), the remaining parts are represented with only the connection parameters (usually $1 \sim 3$D).

The solver handles both the simulation (4.2.4) and the pre-checks (4.2.4, 4.2.4) that check whether the `connect()` operations can be physically realized. We describe the pre-checks before we discussing the simulation of the assembly under gravity.

Verify Connect() Two potential issues can arise when a connection is being made between two parts. First, a connection cannot be made between two primitives that cannot be joined according to the connectivity table (Table 4.2), such as a rod to another rod, or when they are two *closed primitives*.

Second, the solver must check whether the *available critical dimension* of a primitive is sufficient for what is needed for a new connection. Based on the primitives' critical dimensions, we add constraints to the parameters of the connector primitive that has multiple connections. For example, when two hooks connect to the same rod, two sets of parameters that decide where along the rod the hooks connect are created. Suppose that the hooks each have widths w_1, w_2, the rod has length l, and the two connection parameters indicating the position of the hooks along the length of the rod are $t_1, t_2 \in [0, 1]$. Then, this "multi-connection" constraint $|t_1 - t_2| \cdot l \geq \frac{w_1+w_2}{2}$ is created and included in the solving process. We represent this constraint as a soft penalty, as follows:

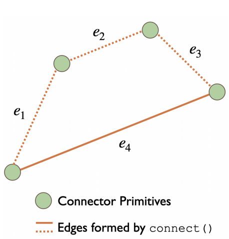
$$C_{m,f} = 0 \text{ if } f \geq 0,\ C_{m,f} = f^2 \text{ if } f < 0,$$

where $f = |t_1 - t_2| \cdot l - \frac{w_1+w_2}{2}$. We use the symbol C_m to represent the sum of all multi-connection constraint penalties.

Additionally Verify Connect() that Creates Cycles A `connect` operation creates at least one cycle in the graph representation of the assembly if it is between two parts that are already part of the assembly (Figure 4.21). Such cycles require explicitly modeling constraints over the configurations of the parts being connected. Thus, we must also check whether we can find a set of values for the degrees of freedom that satisfy these constraints.

For a cycle, we model six constraints measuring the failure of the connector frames on the two `connect`ed primitives to match each other. An assembly with n cycles is feasible if valid values of the connection parameters exist along the cycles that satisfy $6n$ equality constraints of the form $f_i(\mathbf{x}) = \vec{0}, i \in [1..n]$, where $\mathbf{x}$ is a vector of all the DoFs in the assembly and $f_i(\mathbf{x}) \in \mathbb{R}^6$ measures the failure of the ith cycle to close up. We minimize the sum of constraint residuals $C(\mathbf{x}) = \sum_{i=1}^{n} \|f_i(\mathbf{x})\|^2$ subject to bound constraints on the DoFs, $\mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max}$. We use the Powell method [195] to minimize C and, if the solver succeeds in finding parameters with $C(\mathbf{x}) \leq 10^{-6}$, the assembly is considered feasible (and thus the `connect` successful). Since the success of the minimization depends on parameter initializations and can get stuck in local minima, we repeat the optimization T times starting from different random initial guesses. We terminate early if a solution is found. We observed that $T = 16$ works well in practice.

Geometric Quick Reject Before we run a full optimization to find a system configuration that satisfies the connection constraints, we also utilize some precomputed information about the parts and primitives to perform a quick geometric check.



Our geometric check uses the triangle inequality: k line segments of length $\ell_1 \geq \ell_2 \geq \cdots \geq \ell_k$ cannot be arranged into a closed loop in 3D unless $\ell_1 \leq \sum_{i=2}^{k} \ell_i$. To apply this principle to our problem, we note that since each part i in a part cycle is rigid, we can bound the Euclidean distance $e_i \in [e_i^-, e_i^+]$ between the point where part i connects to parts $i-1$ and $i+1$. Because $[e_i^-, e_i^+]$ depend only on the geometry of the part and its two connectors involved in the cycle, not on the connection parameters, we can precompute these bounds for all the parts defined in our Annotated Object Library.

Consider the inset figure representing a design that has a cycle of 4 parts. To close the cycle, the following linear program over the distances e_i between connectors must be feasible:

$$\min_{e_i} 1 \quad \text{s.t.} \quad \begin{bmatrix} 1 & -1 & -1 & -1 \\ -1 & 1 & -1 & -1 \\ -1 & -1 & 1 & -1 \\ -1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} e_1 \\ e_2 \\ e_3 \\ e_4 \end{bmatrix} \geq \vec{0} \tag{4.1}$$
$$e_i^- \leq e_i \leq e_i^+, \quad i \in [1..4].$$

Checking the existence of a set of distances e_i satisfying the bound constraints and triangle inequality then amounts to checking the feasibility of a set of linear inequality constraints. This can be solved in milliseconds by standard Python libraries, quickly rejecting impossible `connect` operations.

Stall Prevention When we run the optimization, we put in measures for stall prevention. To halt optimization of C when the solver stalls, we pass a custom callback function to `scipy.optimize` that performs linear regression on $C(\mathbf{x}_i)$ for a sliding window of the last ten DoF iterates $\mathbf{x}_i$. We abort the optimization in failure if the slope of the fit line is less than 0.1 (i.e., the optimizer is not making much progress). This strategy gained us an additional 1.4x speedup on average for examples with cycles in Figure 4.19.

Solving the Assembly After a valid assembly is constructed in FabHaL, the user can invoke the solver to find the values for the degrees of freedom in the system that bring the target part as close as possible to its specified configuration while being in static equilibrium under gravity and respecting all cycle-closure and critical-dimension constraints.

This is a constrained optimization: we want to minimize the user objective subject to the balance of forces and torques on each non-environment part. Early experiments revealed that black-box nonlinear optimization was prohibitively slow at solving this problem and often failed to converge to a feasible local minimum. Therefore, we propose instead a two-step solver that first minimizes the user objective subject to all constraints being satisfied and then uses the optimized configuration as an initial guess for a simulation that relaxes the assembly to static equilibrium.

Step 1: Minimizing the User Objective We use the Powell [195] method to find a feasible configuration of the assembly that minimizes the user objective:

$$\mathbf{x}_{\text{feas}} = \arg\min_{\mathbf{x}} f_{\text{obj}}(\mathbf{x}) + \sigma(C_m(\mathbf{x}) + C(\mathbf{x})) \quad \text{s.t.} \quad \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max},$$

where $C_m(\mathbf{x})$ are the multi-connection constraints described in Section 4.2.4, $C(\mathbf{x})$ are the cycle-closure constraints in Section 4.2.4, and σ is a penalty parameter starting from $\sigma = 100$. If after optimization the constraint residual is not below 10^{-6}, we double σ and repeat the optimization, using $\mathbf{x}_{\text{feas}}$ as the initial guess. We repeat this process up to 5 times, which is usually sufficient to find $\mathbf{x}_{\text{feas}}$. If the constraint residual is still not below 10^{-6} after 5 times, we pass the best configuration found to the second step.

Step 2: Relaxing under Gravity We use a physics solver to relax the assembly to an equilibrium state under its self-load, starting from the guess $\mathbf{x}_{\text{feas}}$. Let $q_i \in SE(3)$ represent the configuration of the ith part and $\mathbf{q} = \{q_i\}_{i=1}^{n}$ represent the configuration vector of the entire assembly. For an assembly with c total pairs of primitives connected, let $g_j(\mathbf{q}, \mathbf{x}) \in \mathbb{R}^6$ for $j = 1, \ldots, c$ be constraint functions encoding that each pair of primitives are connected with connection parameters $\mathbf{x}$.

To relax the assembly under gravity, we solve

$$\arg\min_{\mathbf{q},\mathbf{x}} E(\mathbf{q}, \mathbf{x}) \quad \text{s.t.} \quad \mathbf{x}_{\min} \leq \mathbf{x} \leq \mathbf{x}_{\max} \tag{4.2}$$

$$E(\mathbf{q}, \mathbf{x}) = \sum_i P_i(\mathbf{q}) + \sigma \sum_{j=1}^{c} \|g_j(\mathbf{q}, \mathbf{x})\|^2,$$

where $P_i(\mathbf{q})$ measures part i's gravitational potential energy and σ is a penalty parameter enforcing that connectors stay attached: we use $\sigma = 100$. We optimize Equation (4.2) using an active-set Newton's method [139].

To demonstrate the two-step process, we take the hack design from Figure 4.21 as an example, which hangs a soap bottle from a rod using eyehooks and a basket. Both programs visualized in Figure 4.23 use the same target configuration specification for the soap bottle; thus, after the first step, the soap bottle is in a configuration that is closest to the target configuration. However, after the second step of relaxing under gravity, without a second eyehook to balance, the top row's design falls under gravity into a less desirable configuration compared to the bottom row's design.

During the physics relaxation, we also predict whether the assembly will fall apart due to connectors slipping off each other. To perform this analysis, we annotate each connection parameter for each primitive in our library with one of three tags:

- `UNBOUNDED` parameters are periodic and should be allowed to "wrap around" from $x_{\max}$ to $x_{\min}$ during optimization. For example, for a ring that can rotate 360 degrees, the angle parameter specifying the rotation of the ring about its central axis is `UNBOUNDED`.
- `BOUNDED_AND_CLAMPED` parameters are used if the geometry of the primitive prevents the parameter from ever leaving the interval $[x_{\min}, x_{\max}]$. The position parameter of a rod along the bottom of a clothes hanger is an example of this parameter.
- `BOUNDED_AND_OPEN` parameters are used if exceeding the bounds of the parameter would cause the assembly to fall apart. The position parameter of a dowel rod, for example, is `BOUNDED_AND_OPEN`: hooks or rings that slide past the end of the dowel rod fall off the assembly.

At the end of optimization, for each `BOUNDED_AND_OPEN` parameter i, we check whether x_i is in the inequality constraint active set, i.e., whether x_i is equal to its maximum or minimum allowed value, and if so,

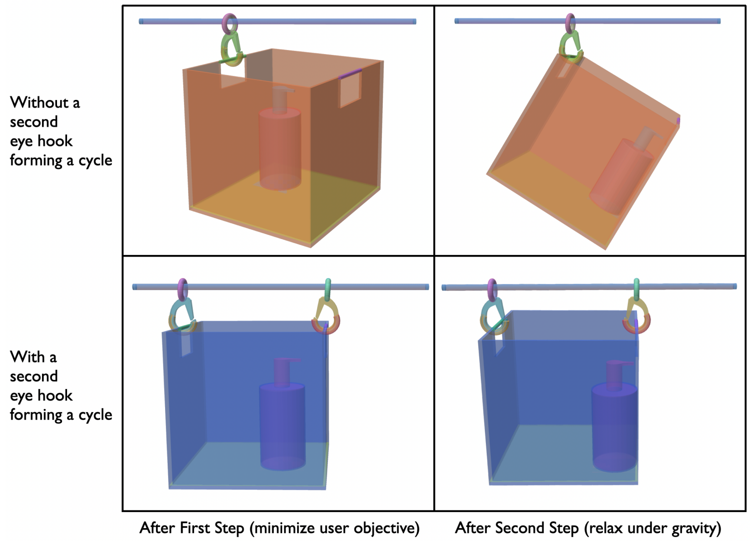


Figure 4.23: The top row shows the hack design without a second eyehook to balance the basket, and the bottom row shows the hack design with the second eyehook. The first column shows the intermediate results after first running the user objective minimization, and the second column shows the resulting configuration after the second step is run.

Figure 4.24: "Demo".

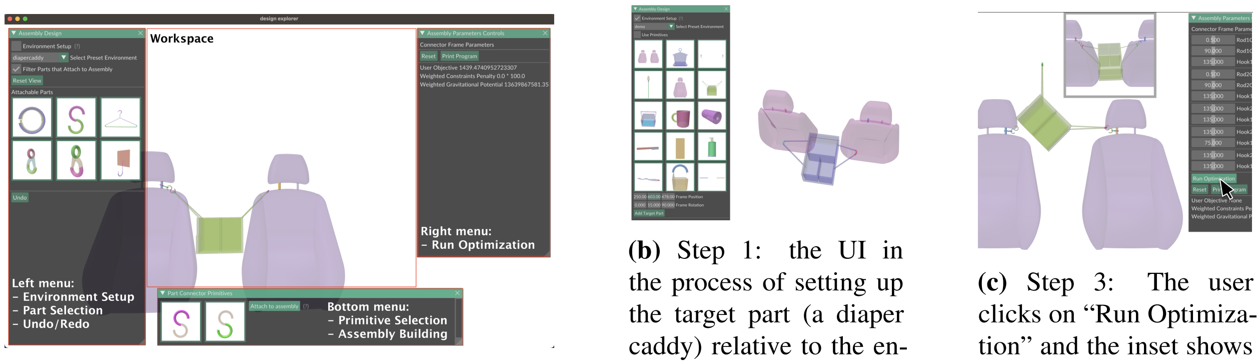
whether $\nabla_{x_i} E$ points away from x_i's feasible interval. If so, we report to the user that the assembly falls apart. A hack assembly might have many different equilibrium states under gravity; our method above finds just one of them. For instance in the "Demo" assembly (inset Figure 4.24), the S-hook and ring could slide to either end of the hanger's rod depending on which end $\mathbf{x}_{\text{feas}}$ encodes they are closer to.

4.2.5 The FabHacks Interface and User Workflow

This section describes how users create FabHaL programs with the FabHacks UI. As introduced in Section 4.2.4, to construct a hack design, users first specify a starting environment with `start_with` and a target part's configuration relative to the environment with `end_with`, and then connect connector primitives on two parts with `connect`. Then, they can use `solve` to check design validity and solve for the configuration of their design under gravity. Based on feedback about whether a connection is valid and the visual feedback shown in the UI, users can choose to iterate on their design, as needed.

The UI consists of the *workspace region* and *three menus* (see Figure 4.25a). The left menu helps users during environment setup and for selecting parts to use in assembly design; the parts shown here are all from the "Annotated Object Library." The bottom menu helps users choose connector primitives of the selected part, and this is where the buttons for constructing the assembly appear. The right menu helps users solve for the assembly's final configuration.

We design the UI interactions to roughly correspond to the program construction process.



(a) The user interface.

(b) Step 1: the UI in the process of setting up the target part (a diaper caddy) relative to the environment (car seats).

(c) Step 3: The user clicks on "Run Optimization" and the inset shows the solved configuration.

Figure 4.25: Left: a screenshot of the user interface. Center: example interactions for Step 1. Right: example interactions for Step 3.

Step 1: Environment Setup Users start by setting up the environments where this hack will be situated. Taking the diaper caddy hanging hack (Figure 4.19f) as an example, in Figure 4.25b, we (as users) have already added the car seats as the starting environment and are in the process of specifying the configuration of the target part (diaper caddy) relative to the environment. The desired configurations (position and orientation) of the environment and target part can be changed with sliders. This completes the environment setup, which corresponds to `start_with` and `end_with` in the program.

Step 2: Assembly Design Next, users construct the assembly by specifying which connections to make. They can either select a part from the left menu to connect it to the assembly or select two connector primitives already in the assembly and specify that they should be connected. As defined in our DSL, `connect` might introduce unsatisfiable constraints and thus need to be verified. We provide two-way filtering based on the connectivity table (Table 4.2) to skip some pre-checks. For example, if a hook is selected in the menu, then only hook, ring, rod, and tube primitives will be enabled for selection in the workspace, and vice versa for a hook clicked on in the workspace. If a connection cannot be made because of failed pre-checks or because the solver cannot find a valid set of parameters that satisfy the constraints, specific feedback is provided to users. For example, if the connection to be made introduces a cycle but the cycle cannot be formed according to the geometric quick reject, the feedback will remind users that the two primitives might be too far away to be connected.

Step 3: Solving After the environment and the target part are fully connected, the assembly is considered "valid." Users can then invoke `solve` with the button "Run Optimization," and the solver positions the parts such that changes in the configuration of the target part are minimized and the assembly is stable under gravity subject to any constraints in the design (Figure 4.25c). We note that this is not an interactive-rate step because the full physics-based solving can take up to a few minutes for complicated assemblies. More details on the runtime can be found in Table A.5 in Appendix A.4.

After users see the visual or textual feedback on their design, they can choose to continue modifying it either with some backtracking via undo and redo buttons or by simply adding more parts. They then re-run the solve to view the updated design. For example, in the case of an unstable design, the solver would return feedback that one connection will fall apart under gravity, and the user might choose to modify that specific connection. In case the solver fails to solve (which happens rarely, as shown in Appendix A.4), the number of random initial guesses to try could be increased in the UI.

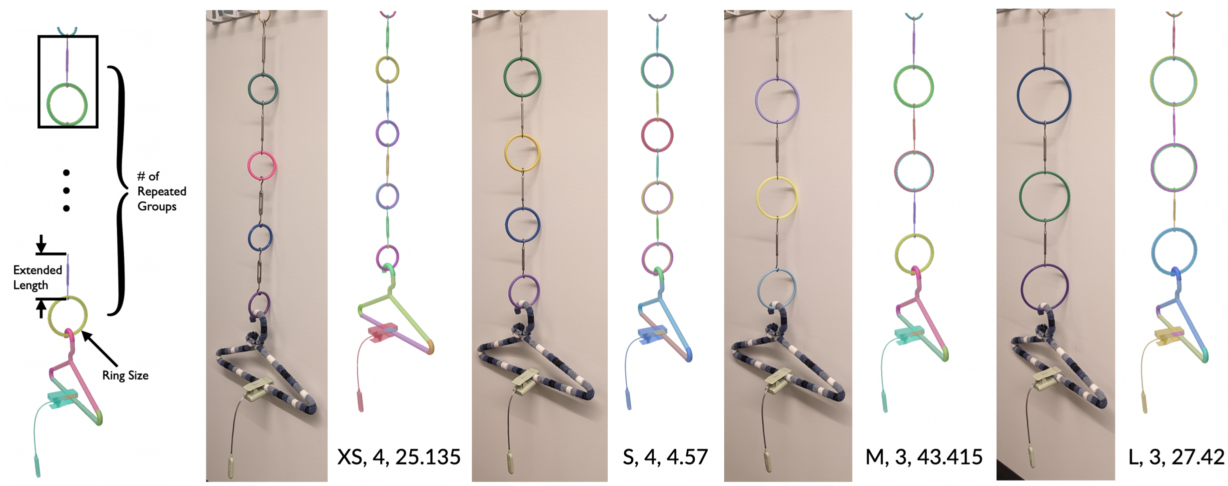


Figure 4.26: The parametrized assembly design and its four variations that most closely match the desired target part configuration given different ring sizes, with the photo (left) and simulated result (right). The parameter combination is indicated below the simulated result.

4.2.6 Evaluation of FabHacks

We now discuss implementation details and how we evaluated our system on both direct programming with examples and programming via an interface with a user study.

Implementation

We implemented FabHaL as a shallowly embedded DSL with Python, i.e., it is embedded in the host language Python without its own abstract syntax tree. This allows the DSL to be used as a Python library and have access to common control structures from the host language for straightforward programmatic design generation. Python as the host language also facilitates easy integration with existing optimization and geometry processing libraries in our solver implementation [195, 165, 91]. The UI is also implemented in Python using polyscope [165] with extended features from imgui.

Examples from Direct Programming and Programmatic Generation

When the user has an overall idea of the hack they want to create, they can directly code their design and virtually test it. Figures 4.15 and 4.19 show some example hacks created via direct programming.

As a DSL, FabHaL also lends itself well to programmatic generation of families of programs. This is helpful when the user knows roughly what parts to use but is not sure of how many. They can generate parametrized designs using the host language features, such as conditionals and loops, and use the solver to find the set of parameters from hundreds of variations that let the program best satisfy the given target part configuration.

As an example, suppose we are preparing for a trip to a summer camp with bunk beds. We would like to hang a clippable reading light at a certain distance from a hook on the top bunk bed so that it is sufficiently far away to not affect others in the same space and we can also reach the light's switch easily. We have a rough idea to use a hanger, extendable M4 turnbuckles, and rings of different sizes and chain them together into a fixture hack for this scenario. The parametrized design consists of a chain of n turnbuckle-ring pairs, with each turnbuckle extended by l millimeters and each ring of radius X (see Figure 4.26, leftmost). With this parametrization, we can programmatically generate a family of programs. If we already know the desired length and the size of rings that we have, we can use the solver to find the best parameters of n and l for a given ring size. Figure 4.26 shows four designs that match the target configuration, each corresponding to the four ring sizes ($X \in \{XS, S, M, L\}$) and selected from the 80 program variations with $n \in [1..4]$, $l \in [0, 45.7]$ (discretized into 20 values).

User Study with FabHacks Interface

Users with minimal coding experience can create designs more interactively with our UI. We evaluate how useful our system is for hack designs through a user study with ten participants. Participant ages ranged from 18 to 34, with CAD experience ranging from none to greater than 5 years. Participants reported their gender as Male (5), Female (3), Non-Binary (1) and N/A (1). We conducted the study in our lab with a laptop we provided, and each session took about an hour. Audio and screen capture were recorded during the study.

Method After obtaining informed consent, we showed participants a tutorial on how to construct the "Demo" assembly (inset Figure 4.24) in FabHacks, and participants repeated the same steps.

Participants completed an **open-ended design task** using FabHacks where they were asked **to hang a bird feeder from two hooks and think aloud during the process**. The wall hooks (environment) and the bird feeder (target) were given. They had up to 30 minutes to create a design and were asked to come up with additional designs if time remained. We coded all designs as *feasible* or *infeasible* and then grouped them into categories based on similarity.

After the study, we asked participants to **answer three questions**: (1) "Can you tell us up to three things you would like to see us **keep** in the FabHacks tool?" (2) "Can you tell us up to three things you would like to **change** in the FabHacks tool?" (3) "Can you think of a **real-world change you would like to make** to your space in the office or at home that the FabHacks tool could help you with?" We grouped the responses into categories and discussed them until we reached a consensus.

Results and Discussion Overall, our study shows that FabHacks is an efficient and intuitive way to construct hacks.

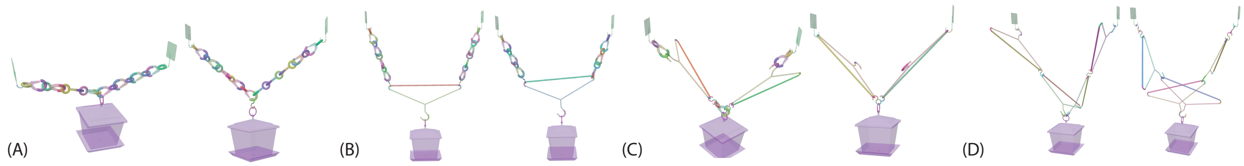


Figure 4.27: Examples of each of the 4 common design strategies (A-D) found by participants.

The 10 participants created 25 feasible designs, each unique though many used similar strategies. Twenty-three of these belong to one of four common strategies: (A) constructing symmetric chains of small objects to anchor the birdfeeder between the two hooks (7 instances), (B) constructing two short chains, hanging a coat hanger upside-down between them, and dangling the birdfeeder from the hook of the coat hanger (6 instances), (C) hanging a coat hanger from each wall hook and anchoring the bird feeder where they meet (8 instances), and (D) chaining two coat hangers from each wall hook and connecting the birdfeeder in the middle of them (2 instances). While several participants discovered each pattern, no two were identical; they chose different types or numbers of parts to achieve similar construction or connected the same parts in different ways.

Looking at participants' answers to our three questions, we saw several important themes arise.

Question 1: FabHacks Keepers. First, multiple participants liked how "intuitive" the FabHacks interface was and praised its physics solver. One participant praised the "real-time realistic feedback" on connections, and another praised the "simplicity" of making connections in FabHacks.

Question 2: FabHacks Changes. At the same time, participants noted areas for improvement. For example, multiple users mentioned that "not knowing the reason a [connection] is failing [when validation is run] can be frustrating" and asked for a wider variety of undo and delete operations (a simple feature to add). Participants also made suggestions such as: having a constraint on the number of available pieces; better support for orienting, panning, and zooming; a tree diagram showing the connections; and better feedback about what is selected.

These critiques generally represent opportunities for improved user experience design rather than fundamental flaws with the mental model required to use our tool. For example, it would be possible to tell the user more specifically which part had a geometric flaw (e.g., being not long enough) and caused a connection to fail, or to visualize the configuration found by the solver with the failed connections highlighted.

Question 3: Real-world Use Scenarios. We also found that 6/10 participants had concrete ideas for how they would use FabHacks in their everyday lives, from a tree swing to outdoor lights to wall hangers to hang decorations or photos. Of four participants who did not see a use for FabHacks, one felt that the library needed to be expanded and account for things like weight because otherwise they would prefer testing the design directly; one felt they could more easily make a plan in their head; and two did not have an idea for how to use it.

Although participant comments suggest that there is room for improvement, the preceding feedback mostly focuses on things that can be solved with a larger library and iteration on the user experience. Future work could explore adding physical properties like weight and center of mass to our physics solver or letting users choose other materials for their parts.

4.2.7 Limitations and Future Work

This work introduces a design system that helps users create fixture hacks built out of household items. Our solution features a new solver-aided DSL, FabHaL, which was inspired by our analysis of a collection of hacks. Our study showed that FabHacks can support end-user construction of hack assemblies and is intuitive to use. Participants also identified potential opportunities for using FabHacks in everyday life, suggesting that the ideas presented in this work could inspire a new age of sustainable DIY design.

Our work has some limitations. First, it only considers rigid unmodified parts and makes simplifying assumptions on the part interactions. An important future direction is to extend the proposed abstractions to handle any hacks using soft parts, more complicated parts (e.g., with shifting centers of mass), or examples where the part shapes can be altered during assembly, such as a piece of wood that can be cut to size. For example, our parametric connections can be expanded to accommodate additional degrees of freedom, allowing for the representation of deformable objects or items that can change dimensions when cut.

More physical solvers could also be incorporated to handle deformable shapes and more complex part-part interactions. For example, more advanced methods could be employed to determine the physical compatibility of parts instead of relying on approximations through closed primitives and critical dimensions.

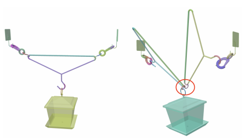
Our user study was designed to verify the usability of our system for creating valid hack designs without physical prototyping. Although we did not ask participants to build the hacks they designed, we retrospectively built five designs for which we had enough parts and verified their physical stability. In the future, understanding how users physically prototype—as well as how hacks get assembled, disassembled, and actually used—would greatly inform how to improve the current system.

Our system also presents opportunities for future work. First, although quantifying the environmental impact of designs was not the focus of this work, it would be valuable to develop approximate metrics for designs that can be created from existing objects, leveraging surrogate attributes like weights or utilizing AI-assisted searches. Knowing a quantified EI number could help users recognize their positive (or potentially negative) impact, which might affect or inform their choices.

It would be also interesting to explore automated recognition and fitting of connection primitives given a 3D model of a part. We can take this a step further by automatically adding a part to the library from LIDAR data or multiple images of an object, which would enrich the modeling power of the system and help bridge the reality gap.

Another promising opportunity is the complete automation of assembly design. By abstracting out eight common connector primitives and rules on their connection behaviors, our proposed DSL not only supports interactive design but has the potential to facilitate the generation of optimal designs under various objectives because it fundamentally reduces the search space. Automating the design of home hacks is a challenging task because it involves searching through discrete combinations of parts and finding suitable continuous parameters that meet the specifications. Our abstractions enable us to decouple this problem into a program synthesis task nested with continuous optimization, which is performed by our solver. How to make program synthesis techniques usable in this context poses an interesting research problem.

FabHaL as a DSL could also benefit from the recent advances in large language models. Recent experiments that use LLMs for generating [94, 170] or completing programs [149] in various DSLs show promising results. In preliminary experiments, we prompted GPT-4 [143] to design a hack for hanging the birdfeeder with eyehooks, S-hooks, and hangers. While most attempts did not lead to a desirable design, GPT-4 was able to propose valid and near-valid designs (see inset). The inset-left shows a design created by GPT-4 that is very



close to our design (Figure 4.15), and the inset-right shows a design that is not physically valid when evaluated with our solver (the S-hook will disconnect and fall off) but resembles the participant-created designs using strategy C (Figure 4.27). Exploring how to enable LLMs to create physically valid hack designs with FabHaL is an exciting research direction: the underlying solver could become useful in generating feedback based on the solve results to prompt LLMs to fix issues in invalid or undesired designs.

Chapter 5

Sustainability-Constrained Design with Sustainability Feedback

In Chapters 3 and 4, we saw that design systems can be built to either enable a more informed design process, or enable designing under constraints posed by sustainable manufacturing practices. In the real world, however, designers often need to deal with constraints while considering sustainability as part of their design goals. In this chapter, we show how we can build design tools for this more complex scenario. Specifically, this chapter introduces a design system that considers constraints from using leftover fabric materials (e.g., inconsistent sizes, large variety of appearances and shapes) while enabling designers to understand how much of the leftovers can be reused throughout the design process. The underlying key insight is still to devise suitable abstractions with domain knowledge, but now we need to consider both from a design perspective (allowing easy design exploration, evaluation of sustainability metrics, higher-level design controls, etc.) and from a sustainable fabrication perspective (encapsulating the relevant constraints).

5.1 Introduction

The process of making textile objects creates fabric leftovers of various colors, sizes, and quality. Since fabric production is a resource- and chemical-intensive process, people try to put any textile waste to a second use before sending them to landfill. In a factory setting, smaller pieces are typically downcycled into filler materials or recycled into fibers for yarns, while larger, higher-quality scraps like faulty-cut panels can be upcycled into new products of smaller sizes [103]. Some fabric recyclers [63, 62, 200] also directly work with designers, apparel manufacturers, and local maker communities to collect any scraps like off-cuts and deadstock fabrics and sell any reusable scraps to give them a second life.

Among the many ways to reuse these fabric scraps, *patchwork* is a popular option. The word itself can refer to the technique of assembling pieces of fabrics with stitching, or the objects produced with that technique [112]. These scrappy-looking textile designs have a long history and are seen around the world. Patchwork was once used when materials like cotton were more rare and valuable. Today, patchwork has

Figure 5.1: We propose ScrapPack, an interactive design tool for reusing scrap fabrics by packing them in a way guided by traditional quilt block designs. The newly made fabrics from the scraps can be used in other crafting projects. Here we show three examples made from patchwork fabrics designed using our system. From left to right: tote bag; box t-shirt (toddler size); skirt (adult size). Each example uses a different quilt block as the patchwork design guide, and the estimated dimensions of the patchwork from our tool are listed.

become more of a style in textiles, and many individual crafters and artists enjoy using scraps (their own or bought from recyclers like FABSCRAP [63] and FABCYCLE [62]) to achieve such a visual style while being environmentally friendly [27, 144].

However, creating a visually attractive and well-constructed patchwork design with fabric scraps is challenging. Fabric scraps often have very different shapes, sizes, and colors, which makes it difficult to manually explore a large space of design possibilities. It is even more challenging if one wants to maximize their scrap material usage and ensure the final patchwork piece is large enough for an upcycling project. Existing patchwork design tools [88, 110] focus on visual aesthetics and design freedom, so the created designs may not be using scraps as effectively or sustainably as possible. To address this, we present ScrapPack, the first material-centric patchwork design tool. ScrapPack starts from the users' available scraps and helps users make the most of them while still supporting creative and visually pleasing designs.

Given the available scraps, putting them together to cover the largest possible 2D region can be seen as a form of 2D bin packing problem, which is well-studied [99, 89]. However, solely focusing on maximizing the final packed area and the scrap material usage through an automatic method could lead to visually incoherent designs because the automatic methods do not take the visual aspect into consideration. Thus, we need a way to bridge user control over the aesthetics of the design and packing optimization.

To address this challenge, we take inspiration from patchwork quilting. Our main insight is that some quilt blocks lend themselves well to efficient and aesthetic packing due to their sequential strip structures. First, traditional quilt blocks often follow visual design principles like balance and harmony which makes them aesthetically pleasing. Second, the strip structures serve as an intermediate design medium, breaking down the patchwork design choices into sequentially added strips that could be laid out in visually meaningful ways. This exposes the high-level design decisions of selecting strips to users while still allowing efficient optimization through mixed integer programming to maximize scrap reuse in each strip. Specifically in this work, we use three pieced block patterns made of strips [18]: log cabin, courthouse steps, and rail fence (diagrams in Figures 5.1, 5.5).

In addition, the strip structure promotes constructability of the patchwork design, only involving two main categories of construction: piecing scraps into a strip, and piecing a strip with the patchwork design in progress. To facilitate constructing the designs, ScrapPack also automatically generates fabrication instructions.

We showcase ScrapPack's design capabilities with three upcycling projects created with patchwork fabrics designed using our tool (Figure 5.1). To further evaluate our quilt-block-inspired packing algorithm and the design tool, we conducted technical evaluations of the algorithm (Section 5.6) and a user study with an interactive packing interface (Section 5.7). Participants generally found ScrapPack helpful in using scrap material efficiently and reducing manual effort needed to create the patchwork design, and felt inspired about how the tool could help them reuse their own scraps.

5.2 Background and Related Work

Our work builds upon prior work in computational tools for quilting and material reuse.

5.2.1 Quilting and Patchwork

Quilting is a process that involves designing a layered textile object. Often, three layers are involved: the decorative top layer where design elements are placed, the batting middle layer which creates warmth and volume, and the plainer backing layer [112]. The top layer usually involves the main visual design created through patchwork (piecing multiple fabrics into a single layer through sewing) or appliqué (attaching multiple layers of fabric on top of each other).

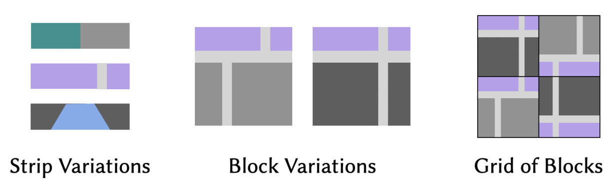


Figure 5.2: Quilt design can be broken down into a hierarchical structure, starting from deciding on fabrics, then individual strips and blocks, and finally the layout of the blocks.

Many quilts are block-based, meaning that a single geometric pattern is repeated in a grid layout (Figure 5.2 right). Common geometric shapes can be found in the block patterns, including rectangles, squares, diamonds, and triangles (Figure 5.2 middle). In this work, we focus on block patterns that mainly consist of rectangles (or squares) as shown in Figures 5.1, 5.5. We refer to these rectangles as "strips," which are composed of smaller fabric pieces and can vary widely (Figure 5.2 left). In this work, we center the design variations around the strips since it provides some level of abstraction and allows both users to still have design freedom and the tool to optimize for scrap material usage.

Quilting involves several different steps, and computational tools have been developed for different steps or with a focus on a few techniques. Igarashi and Mitani [88] focused on the patchwork aspect and created an interactive tool for visualizing patchwork designs. Several works [119, 117, 25] developed algorithms for generating free-motion quilting (or stitching, thread painting) paths based on input portrait images, from parametric designs inspired by other algorithms, or from a sequence to make an animation from quilted frames. Leake et al. [109] studied the piecing technique and developed a mathematical understanding of foundation paper pieceable quilts, which paved the way for a sketch-based design tool [108]. Tools to support the design of other specific forms of quilts, such as bargello quilts [37] and improvisational quilts [111], have also been explored. Recent work has explored ways to support patchwork design using scraps. Most similar to our work, ScrapMap [110] supports block-based scrap quilt design using traditional quilt blocks. While their work focuses on optimizing color layouts for multiple-block quilts, it does not consider scrap reuse efficiency. We focus on optimizing material reuse so that we can create large sheets of patchwork fabric for use in many different applications.

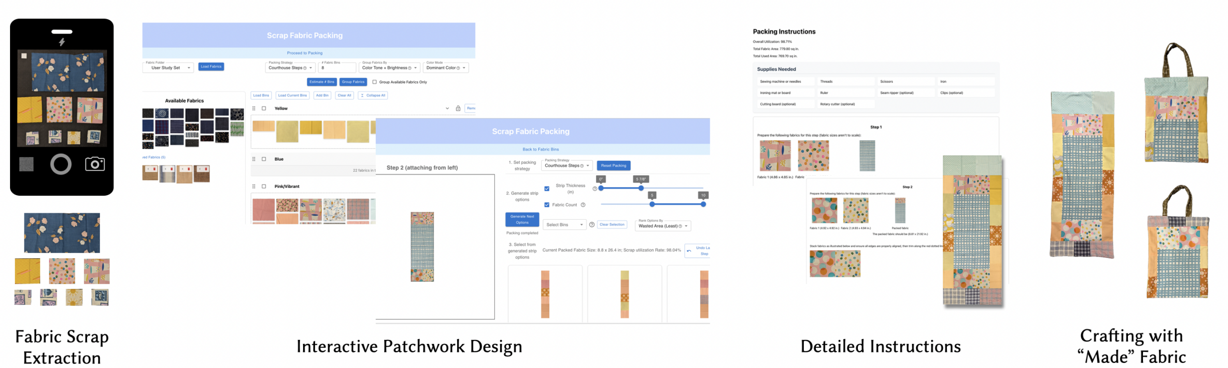


Figure 5.3: ScrapPack's workflow. The user begins by uploading photos of their scraps. Our tool automatically extracts the scrap boundaries and sizes. They then use the UI to sort their fabrics into bins and select strip options. The tool automatically generates detailed instructions, and then the user can sew the resulting patchwork sheet and create any item they wish with it, such as a tote bag.

5.2.2 Collage Design

Closely related to quilting is the craft domain of collage. Collage is a form of visual arts that involves gluing elements together [113]. The concept of collage has been widely employed and evolved in domains such as visual arts (paintings, sculptures, etc.), architectural design, and landscape design [2]. Quilting can also be thought of as a kind of collage of fabrics, and in fact, there are glue-based quilting techniques such as fusible appliqué. Paper collages and photo/picture collages are the most similar to fabric collages. Several computational methods [199, 158, 9, 70, 86] have been developed for photo collage, and the focus is to figure out an optimal placement and transform (scaling and rotation) of images so that the collage looks like some other input imagery or shape. Fabric collages also need to consider visual features when placing elements, but creating collages using scrap materials is constrained by the actual sizes of scrap pieces. Shinjo et al. [130] created a workflow that takes in an image and converts it to a mosaic art pattern that can be recreated using scraps by tucking scraps onto a laser-cut board. Our work focuses on helping users stay aware of their scrap materials so that they can reuse as much material as possible while still having some level of artistic control over the fabric collage design. We also provide sewing assembly instructions so that the outputs can be used as material sheets for new objects.

5.2.3 Minimizing Waste through Packing

Bin packing and cutting problems have been widely studied in industries like fashion, logistics, and manufacturing where it is desirable to have zero/minimal-waste designs and packaging to save space and materials [216, 68, 57]. Packing has also been used towards creative applications, optimizing for artistic goals in the final packing [156]. The traditional bin packing problem involves packing rectangles into a fixed-size rectangular bin, and is known to be NP-Hard. Variations of the problem explore different parameters for (1) the dimensionality (usually 2D, but there are 1D and higher dimensional variants); (2) the kinds of shapes to pack (orthogonal, irregular but polygonal, or any general shapes); (3) the optimization goal (e.g., minimize number of bins required, minimize material/space waste); and (4) the constraints (e.g., the guillotine constraint which means only edge-to-edge cuts are allowed [150]).

Usually, bin packing problems are solved by heuristic-based optimization methods (for a thorough review of recent exact methods and relaxations, please see [89]). Our packing approach differs from typical bin packing problems in several key ways. First, we do not pack towards a fixed-size bin. Second, we require the packing to have no holes while bin packing/cutting does not enforce that there are no gaps between the shapes. Third, we want to maximize the amount of scrap fabrics used. Lastly, we allow scraps to be cut during packing and keep track of the cut-off shapes if their dimensions are larger than 1 inch.

A related problem to bin packing is the polygon covering problem, where the goal is to cover a polygonal region, often with constraints such as rectilinearity, using the minimum number of simple geometric shapes like rectangles [76]. This problem allows arbitrary shape overlap and produces a no-gap covering. However, the shapes can take any size needed to cover the input shape. Since we are working with physical materials, we cannot increase the size of shapes beyond the available fabric.

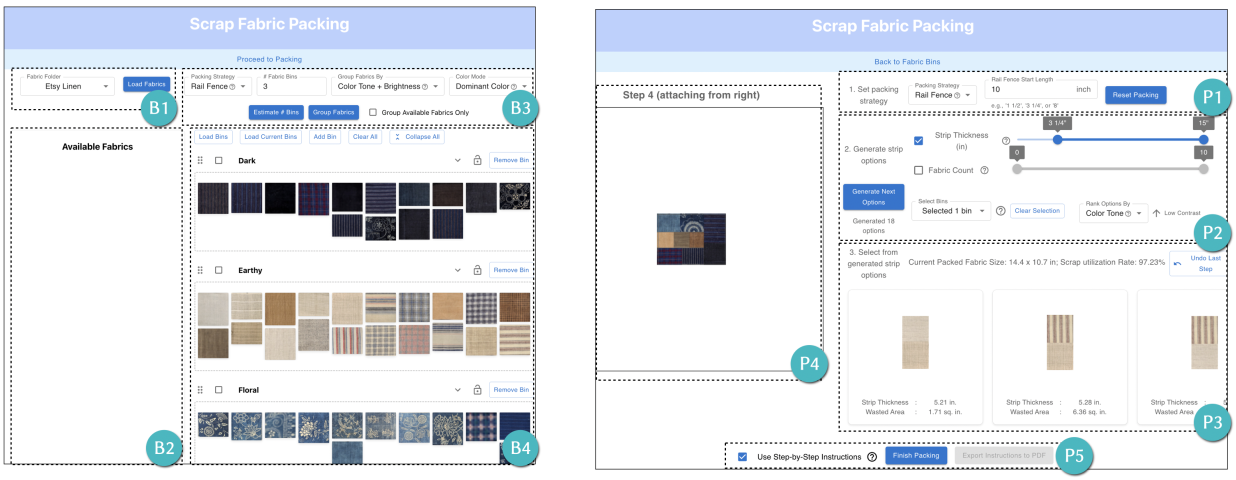
To summarize, in this work, we develop a quilt-block-inspired interactive packing algorithm with the physical constraints posed by fabric scraps, and the requirement that the packing should have no holes. The structure and visual appearance of the packing result is inspired by three traditional quilt blocks. Our approach presents a novel packing algorithm that considers the special constraints imposed by working with fabric scraps intended to be sewn into sheets for reuse.

5.3 System Overview

In this work, we create a design tool ScrapPack for users to interactively design a "new" fabric from their fabric scraps (Figure 5.3).

We chose to separate the patchwork design process into the binning step and the packing step (as seen in

Figure 5.4). This was a design decision informed by a formative study with five quilters who have used scraps and books on using scraps [17, 53, 36, 77, 182, 189], through which we found that scraps are usually sorted by fabric properties such as color or print, size or shape, and material. Strip designs will then be generated from within the sorted fabric bins.



(a) Binning UI. B1: loading fabric scrap sets. B2: the area displaying available scraps to be binned (empty means all are sorted into the bins). B3: available controls for automatic binning. B4: the area displaying fabric bins.

(b) Packing UI. P1: packing strategy selection and additional parameters if any. P2: controls that influence strip option generation. P3: generated strip options and information about current packing. P4: the patchwork design in progress. P5: the area displaying instructions.

Figure 5.4: ScrapPack's UI screenshots. The example design here is discussed in more detail in Section 5.3.2.

5.3.1 Design Workflow

The user starts with uploading photos of fabric scraps, and we provide a semi-automatic pipeline to extract the images (Section 5.4.1). Then, the user will group fabrics into bins based on different criteria like color tone, color brightness, or patterns (solids or prints). Our tool provides automatic sorting based on color properties and also allows manual drag and drop to sort the fabrics (Section 5.4.2).

Then, the user selects a packing strategy inspired by different quilt blocks (see Figure 5.5) and enters the packing stage. During packing, the user starts with a scrap piece (or nothing in the case of Rail Fence) and chooses a strip option to pack. When generating the next set of options to choose from, users can express their preferences in terms of ranking criteria or filtering constraints (Section 5.4.3). For a generated strip option, the user can click on that option to open a reorder panel that allows drag-and-drop to reorder the fabrics and obtain the permutation they like.

Finally, detailed instructions can be generated for how to construct the user's design (Section 5.4.4). Our system takes into account seam allowances so that the design can be successfully pieced. After constructing this patchwork fabric, the user could use it for other projects like home accessories and garments.

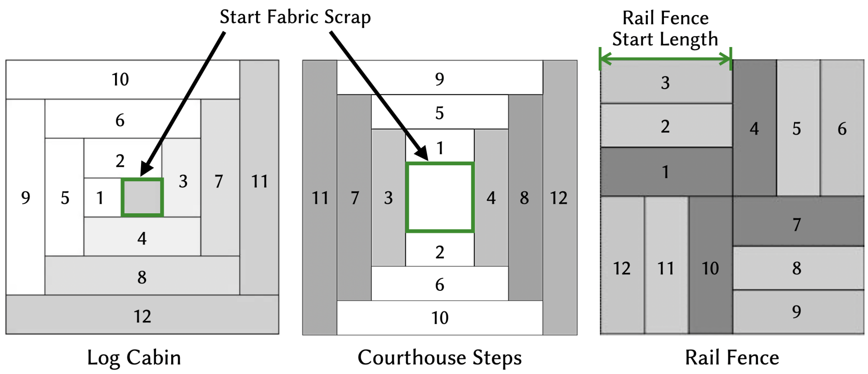


Figure 5.5: Three quilt blocks we take inspiration from and their corresponding strip packing sequences. Based on which quilt block to use as a guide, the strip options generated will be matched to a different target length (see Target Length Constraint 2). The Log Cabin design will match to the left side, the top side, the right side, and the bottom side, and then repeat the sequence. The Courthouse Steps design will match to the top, bottom, left, and right sides and then repeat. The Rail Fence design requires a start length parameter L_S as the target for the first strip and does not continue beyond the 12-strip pattern.

5.3.2 Design Tool Walkthrough

We will now walk through how to use ScrapPack with an example design of a scrappy patchwork mat (Figure 5.6).

Figure 5.6: Scrappy mat. Left: patchwork design (Rail Fence) and estimated size. Right: photo of the mat.

We started with uploading photos of a bag of linen and woven fabric scraps bought from an Etsy shop and processed them into individual fabric images. The image set was loaded into the tool. Then we grouped the fabrics into three bins by “Color Tone + Brightness” and manually adjusted the bins to get the three bins as shown in Figure 5.4a: darker fabrics (renamed to “Dark”), the earthy color tones (renamed to “Earthy”), and the floral patterns and blue-ish fabrics (renamed to “Floral”).

In this design, we chose the Rail Fence packing strategy. Since we wanted to create a mat, we set the start length of the first sub-block to be 10 inches. For each sub-block, we selected one strip from the Dark bin, one from the Earthy bin, and lastly one from the Floral bin. During the packing process, we made sure that each

Table 5.1: Comparison of computed and measured dimensions with error percentages. Fabricated examples were close to the tool's predicted dimensions. Some amount of error or ease is often present in sewing, and these error values are consistent with typical patterns and sewing errors.

Fabricated Item	Computed (W×H)	Measured (W×H)	Width Error (%)	Height Error (%)
Tote Bag Design (Figure 5.1 left)	13.50 × 34.14 in.	13.88 × 35.25 in.	-2.74%	-3.15%
Box T-shirt Design (Figure 5.1 middle)	27.10 × 24.00 in.	27.88 × 24.50 in.	-2.80%	-2.04%
Skirt Design (Figure 5.1 right)	25.50 × 50.20 in.	26.42 × 52.75 in.	-3.71%	-4.83%
Scrappy Mat Design (Figure 5.6)	20.77 × 21.91 in.	21.12 × 21.00 in.	-1.66%	+4.33%
P5's Design (Figure 5.15b)	19.04 × 19.48 in.	19.20 × 19.75 in.	-0.83%	-1.37%

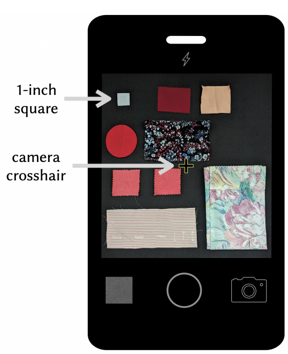
sub-block was roughly similar in size and close to a square by using the Strip Thickness slider to specify thickness constraints. We also used the option sorting functionality; for example, in the current step shown in Figure 5.4b, the strip options were prioritized to have low contrast in color tone within a strip.

Finally, we fabricated the patchwork design following the generated instructions and made it into a mat.

5.4 Methods and Implementation

ScrapPack is implemented using ReactJS for the frontend and Flask in Python for the backend. Next, we explain in detail the methods in each step of the system workflow.

5.4.1 Scrap Fabric Image Extraction



To input scrap fabrics into the system, a photo is taken with the scrap pieces laid down on a flat surface, ideally with the pieces ironed flat. The background surface should contrast with the scrap pieces to make segmentation easier. The camera should be level and right on top of the surface, which can be checked by matching the crosshair on a smartphone's camera. Lastly, each photo should have a 1-inch-square piece of paper placed in the top left corner for calibration purposes. The recommended photo setup is illustrated in the inset figure.

We then use a semi-automatic method to segment out the fabric images from the photos where an automatic segmentation is provided and the results can be manually adjusted in a UI. The approximate dimensions of each scrap fabric can then be calculated with the help of the 1-inch calibration square.

Note that the fabric scrap pieces might not always be flat or rectangular. In this work, we choose to extract the largest rectangular region from each fabric image and only consider rectangular scraps for simplicity.

Calibration Accuracy

To evaluate the accuracy of the calibration method and also the packing algorithm, we measured the dimensions of fabricated designs and compared them with the computed dimensions. As shown in Table 5.1, the relative error is less than 5%. This suggests that our tool provides a good estimate of how large the final design would be, which is important when the user intends to use the final design for a pattern that calls for a specific size.

5.4.2 Fabric Binning

ScrapPack enables users to easily sort fabric scraps by color tone (or hue), color brightness (or value), and color (LAB space color). In addition, the user could select whether they want to use the dominant color (the most common color in the scrap image) or the average color. Users could decide on the number of bins they want to sort the fabrics into, or let the tool suggest a number of bins. We compute the distances of the colors of fabric scrap images following the CIE 1994 textile color difference formula [118], and apply hierarchical clustering with the computed color distances.

As the design continues, fabrics could be used up or trimmed further and we keep track of these changes for the users, logging which fabrics have usable scraps remaining. If a bin runs low on fabrics or the user does not like how the options generated look, they can modify the fabric bins throughout the packing process.

5.4.3 Strip Option Generation

A strip option can be thought of as a combination of different fabrics, or more precisely, fabric images' edges (see Figure 5.7). We treat each scrap fabric as having two edges, one for the width and one for the height. The problem we need to solve here is essentially coming up with sets of edges that sum up to a specific target length (which depends on the guiding quilt block selected by the user). In addition, in order to leave no holes in the packing, we need edge sets with sums no less than the target length, and we also need to consider the user preferences.

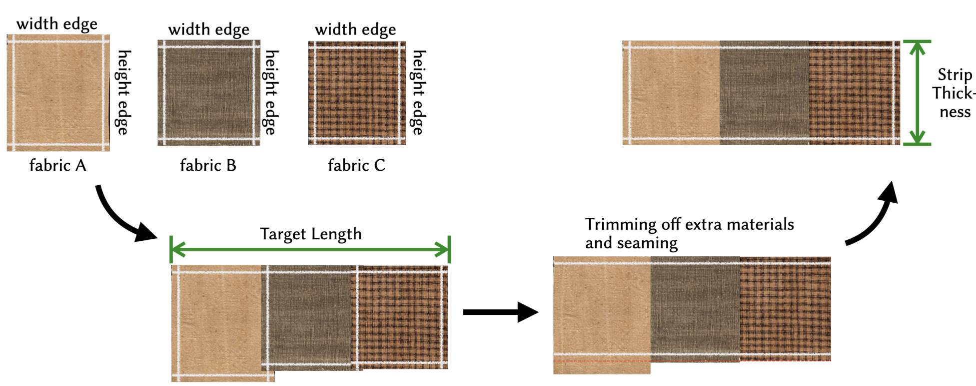


Figure 5.7: An example strip option and its strip thickness illustrated. Three fabrics A, B, C are selected to form this option because their edge lengths sum up to the target length ($w_A + w_B + w_C \geq L_T$). In this case, the strip thickness is the height edge length of fabric C because $h_C = \min(h_A, h_B, h_C)$.

We formulate this edge selection problem as a constrained mixed-integer programming problem and use the Gurobi [75] solver to solve for solutions.

Given a list of n fabrics $\mathcal{F}$, we introduce $2n$ binary variables:

$$\mathcal{E} = \bigcup_{f \in \mathcal{F}} \{e_f^w, e_f^h\}, \quad \text{where } e_f^w, e_f^h \in \{0, 1\}$$

Here, $e_f^w = 1$ indicates that the width edge of fabric f is used to construct the strip, while $e_f^h = 1$ indicates that the height edge is used instead. Selecting the height edge implies that the fabric is rotated by 90 degrees before being attached to the strip. Each fabric can contribute at most one edge, and the selected edge contributes its corresponding length to the total: we denote the width and height of fabric f as w_f and h_f, respectively.

Validity Constraints

We add two kinds of constraints to ensure that the generated strip option is valid:

1. Same Fabric Constraint ($\mathbf{C_{SF}}$):

$$\forall f \in \mathcal{F}, e_f^w + e_f^h \leq 1$$

   which means either width or height edge of the same fabric can be selected but not both;

2. Target Length Constraint ($\mathbf{C_{TL}}$):

$$L_T \leq \sum_{f \in \mathcal{F}} \left(e_f^w w_f + e_f^h h_f \right) \leq L_T + THRESH$$

   where $THRESH$ is a threshold value that allows the generated option to have a bit extra length if the option does not meet the exact target length.

Filter Constraints

Two kinds of constraints for filtering the generated options are available as well: (1) Strip thickness which determines how thick the strip to attach will be. (2) Fabric count which determines how scrappy this strip will look like. Depending on what style the user prefers, they may adjust a slider to indicate that they want thicker strips or thinner ones, or they want more or fewer fabrics in the same option.

1. Strip Thickness ($\mathbf{C_{ST}}$): for each strip option, we consider its thickness to be the minimum length of the unselected edges of the selected fabrics (see definition illustrated in Figure 5.7). We first define a precomputed matrix for all $2n$ edges where

$$W_{ij} = \begin{cases} 1, & \text{if } Length(i) \leq Length(j), \\ 0, & \text{otherwise.} \end{cases}$$

   With this matrix, it is obvious that if an edge i_0 has the smallest length among the selected edges, all

$W_{i_0 j}$ entries should be 1. To facilitate the computation of the minimum length, we add additional binary variables m_f^w, m_f^h for each edge indicating whether its unselected sibling edge is the minimum-length edge. First, only selected edges can be considered for the minimum: $\forall f \in \mathcal{F}, m_f^w \le e_f^w, m_f^h \le e_f^h$. Second, there is only one minimum edge: $\sum_f m_f^w + m_f^h = 1$. Third, the minimum edge should satisfy the precomputed results in the W matrix:

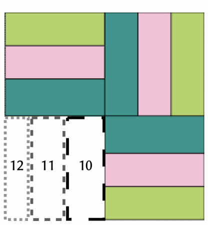
$$\begin{aligned} \forall f, f' \in \mathcal{F},\ f \neq f', W_{f^w f'^w} &\ge m_f^h + e_{f'}^h - 1 \\ W_{f^h f'^h} &\ge m_f^w + e_{f'}^w - 1 \end{aligned}$$

Lastly, we can define the following constraints to ensure the strip thickness is the minimum length of selected fabrics' unselected edges:

$$\begin{aligned} \forall f \in \mathcal{F}, w_f + (1 - m_f^h)M \ge \mathbf{C_{ST}} &\ge w_f - (1 - m_f^h)M \\ h_f + (1 - m_f^w)M \ge \mathbf{C_{ST}} &\ge h_f - (1 - m_f^w)M \end{aligned}$$

where M is a large number (we used $M = 1e6$) so that for example, when the width edge of fabric f is the minimum edge (i.e., $m_f^w = 1, m_f^h = 0$), the second inequality will essentially become $h_f \ge \mathbf{C_{ST}} \ge h_f$ (limiting the $\mathbf{C_{ST}}$ to take on the minimum unselected edge length), while the first inequality is easily satisfiable.

2. Fabric Count ($\mathbf{C_{FC}}$): $\sum_{f \in \mathcal{F}} e_f^w + e_f^h$ computes the number of fabric pieces used in an option.



Additional Constraints for Rail Fence Specifically for the Rail Fence block, in the last sub-block, the three strips are more constrained on the Strip Thickness. Therefore, for the last three packing steps, we encode additional constraints on C_{ST}:

Step 10: $C_{ST} \le L_S$

Step 11: $C_{ST} \le L_S - C_{ST,step10}$

Step 12: $C_{ST} \le L_S - C_{ST,step10} - C_{ST,step11}$

Optimization Objective

Since we want to make sure the packing is as efficient as possible, we always try to minimize the wasted area when generating strip options.

We can approximate the wasted area of a strip option by computing the sum of fabrics beyond the strip thickness as follows:

$$\sum_{f \in \mathcal{F}} \left(e_f^w w_f (h_f - \mathbf{C_{ST}}) + e_f^h h_f (w_f - \mathbf{C_{ST}}) \right) + (S - L_T)\mathbf{C_{ST}}$$

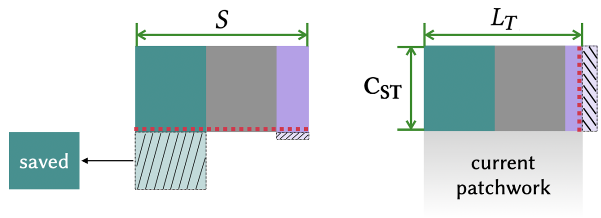


Figure 5.8: Wasted area objective. The left corresponds to the first summation term (portion of fabric scraps that goes beyond the strip thickness), and the right corresponds to the second term (the entire strip's extra portion beyond the target length). This overestimates the wasted area because the cut-off portions that are large enough will get saved instead of being wasted, like the teal piece.

where $S = \sum_{f \in \mathcal{F}} \left(e^w_f w_f + e^h_f h_f \right)$. The first summation term calculates the portion of fabrics that goes beyond the strip thickness, and the second term calculates the portion of the entire strip that goes beyond the target length. This is an overestimation because sometimes the extra fabric beyond the strip thickness may be large enough to be reused and thus would not be considered part of the wasted area (see Figure 5.8).

Ranking Criteria

Lastly, we provide the following ranking criteria for the generated options:

- Wasted Area (Ascending)
- Color Contrast (Ascending or Descending)
- Color Tone Contrast (Ascending or Descending)
- Color Brightness Contrast (Ascending or Descending)

The Wasted Area criterion is defined the same as the optimization objective. The remaining criteria are computed after the strip options are generated. We encode each ranking criterion as a function of the option's fabrics. Without loss of generality, we only describe the criterion that ranks a certain property in *ascending* order (the descending order is the opposite of how that is defined).

1. Wasted Area ($\mathbf{R_{WA}}$): This is the true wasted area computed after the strip option is known. For a given strip option O, the wasted area is computed as

$$\left(\sum_{f \in O} e^w_f w_f \Delta^h_f + e^h_f h_f \Delta^w_f \right) + (S - L_T)\mathbf{C_{ST}}$$

where $S = \sum_{f \in O} \left(e^w_f w_f + e^h_f h_f \right)$,

$$\Delta^w_f = \begin{cases} 0, & \text{if } h_f - \mathbf{C_{ST}} > \text{THRESH} \\ h_f - \mathbf{C_{ST}}, & \text{otherwise} \end{cases},$$

$$\Delta_f^h = \begin{cases} 0, & \text{if } w_f - \mathbf{C_{ST}} > \text{THRESH} \\ w_f - \mathbf{C_{ST}}, & \text{otherwise} \end{cases}.$$

2. Color Contrast ($\mathbf{R_{CC}}$): We precompute the pair-wise color differences between each fabric so that

$$CD_{ff'} = \begin{cases} color\_difference(f, f'), & f \neq f', \\ 0, & \text{otherwise.} \end{cases}$$

   The `color_difference` function is implemented with the `python-colormath` library and computes differences between the two fabric images' dominant color in the LAB color space. For each pair of fabrics within an option O, we compute the sum of the corresponding entries in matrix CD: $\sum_{f \in O} \sum_{f' \in O} CD_{ff'}$.
3. Color Tone Contrast ($\mathbf{R_{CTC}}$): Similar to Color Contrast (Criterion 2), we precompute the color tone (or hue) difference matrix TD and for each pair of fabrics within an option O, this rank is defined as $\sum_{f \in O} \sum_{f' \in O} TD_{ff'}$.
4. Color Brightness Contrast ($\mathbf{R_{CBC}}$): Similar to Color Contrast (Criterion 2), we precompute the color brightness (or value) difference matrix BD and for each pair of fabrics within an option O, this rank is defined as $\sum_{f \in O} \sum_{f' \in O} BD_{ff'}$.

Solver Implementation Details

Since MIP solvers like Gurobi find only one solution at a time, we used a custom solution callback to store any solution found. To ensure that the solutions are different, in the callback function we add new constraints ensuring the next solution found is different from the existing solutions. To keep the UI responsive, we set a time limit for the solver. In general, the solver can find a variety of options (ranging from 0 to 90+) depending on the available fabrics to use and the user-set filters.

5.4.4 Assembly and Instructions

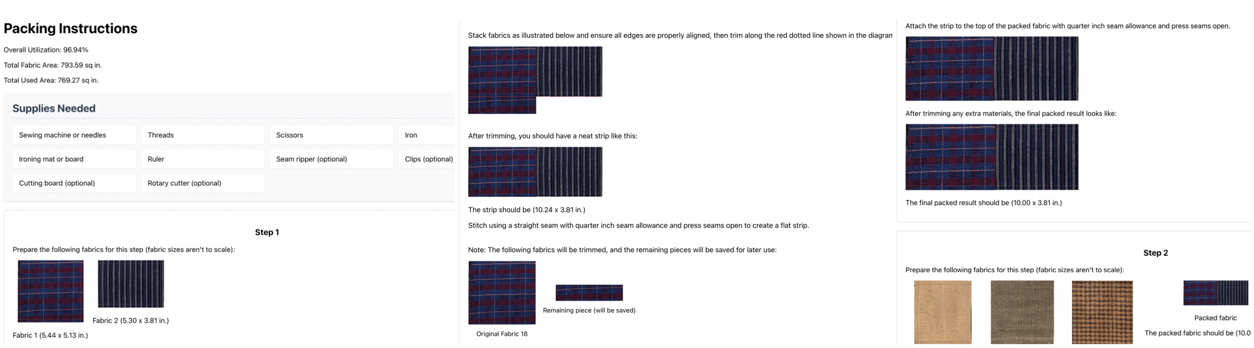


Figure 5.9: Screenshots of the first two steps of the instructions for the scrappy mat design in Figure 5.6 discussed in the walkthrough (Section 5.3.2).

When the user finishes the design, detailed instructions for fabricating the design will be generated and can be exported as a PDF file. During the design process, the took keeps track of each step's selected options and the side it will be attached to, and it puts together the instructions using the saved information. Our system takes into account one-quarter-inch seam allowances for seaming together fabric scraps to ensure the design can be physically assembled. Figure 5.9 shows the first two steps of the instructions for fabricating the scrappy mat design (Figure 5.6).

5.5 Results

To validate ScrapPack's workflow and showcase the potential to design with the tool, we obtained three fabric scrap bags (one from FABSCRAP [63] and two from Etsy) and digitized them with the method explained in Section 5.4.1, and also generated one fabric-image-textured image set using Adobe Stock images and tilable fabric textures [196]. In the rest of this paper, we will refer to them as FABSCRAP, Etsy Cotton, Etsy Linen, and Stock Image Textured.

With ScrapPack, we can easily create design variations through different packing strategies (Figure 5.10a).

Without changing the packing strategy, we can also achieve significantly different designs through the various filters and option ranking methods available to control the final design. For example, the first two columns in Figure 5.10b illustrates the impact of the strip thickness filtering, while Figure 5.10c shows the control provided by one of the option ranking methods.

Fabric bins have a significant impact on the design as well. In Figure 5.10b, the last two columns show that given the same packing strategy and thickness filtering, the designs can still look different due to how the fabrics were binned.

Lastly, we validate our tool's instruction generation by fabricating patchwork designs created with the tool and using them for upcycling projects (Figures 5.1, 5.6).

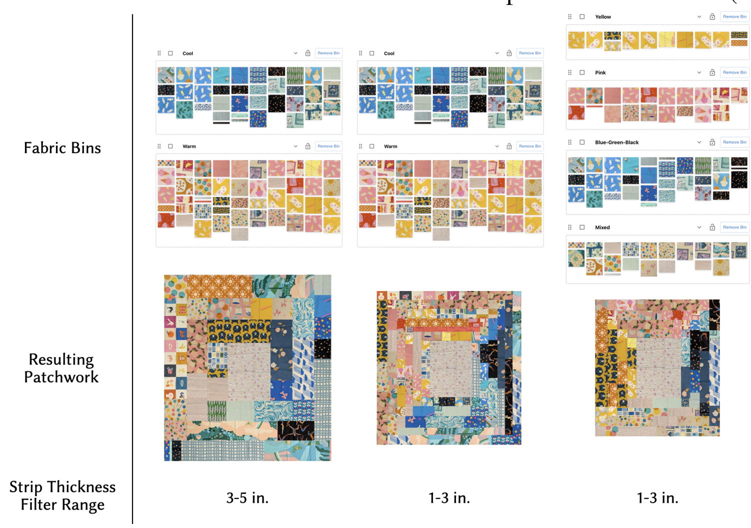
5.6 Technical Evaluation

In this section, we evaluate our proposed packing algorithm by testing it with different distributions of fabric scrap shapes, as well as comparing it with existing bin packing algorithms with open-source implementations.

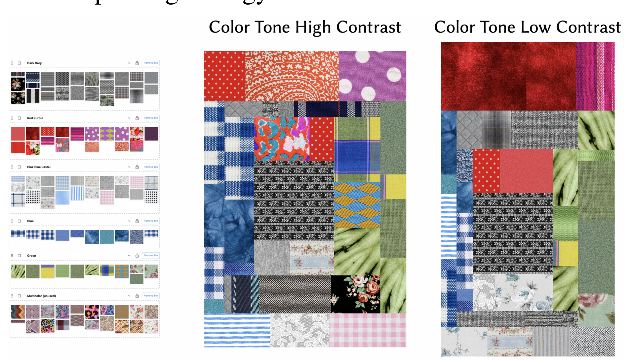
5.6.1 Testing with Different Distributions of Fabric Scraps

In order to understand the differences between the packing strategies and how well they might work with different distributions of fabric scraps, we tested them with the three fabric scrap sets discussed in Section 5.5, and additionally eight generated sets. The generated sets are single-color images and by default, each has a 20% probability of being a square. For each test set, we vary the distribution of fabric scrap sizes and sometimes also the square probability.

(a) Using the Etsy Linen fabrics to create designs with all three packing strategies. Notice that the designs still resemble their quilt block guides and look distinct even when the same set of scraps and the same bins (see Figure 5.4a) are used.



(b) Using the Etsy Cotton fabrics to create designs with Log Cabin. Binning variation: the first two columns use the same bins where the fabrics are grouped into a bin of Cool colors and a bin of Warm colors. The third columns use a 4-bin grouping where fabrics are grouped into “Yellow”, “Pink”, “Blue-Green-Black” and “Mixed”. Filtering variation: the first column uses a strip thickness filter of 3-5 in. while the last two columns use 1-3 in. Both variations bring about visual differences even with the same packing strategy.



(c) Using the Stock Image Textured fabrics to create designs with Courthouse Steps. Ranking variation: by changing how we sort the generated options and prioritizing high contrast or low contrast within the strip, even when the bins are the same, the visual results are conspicuously different.

Figure 5.10: Possible design variations using ScrapPack.

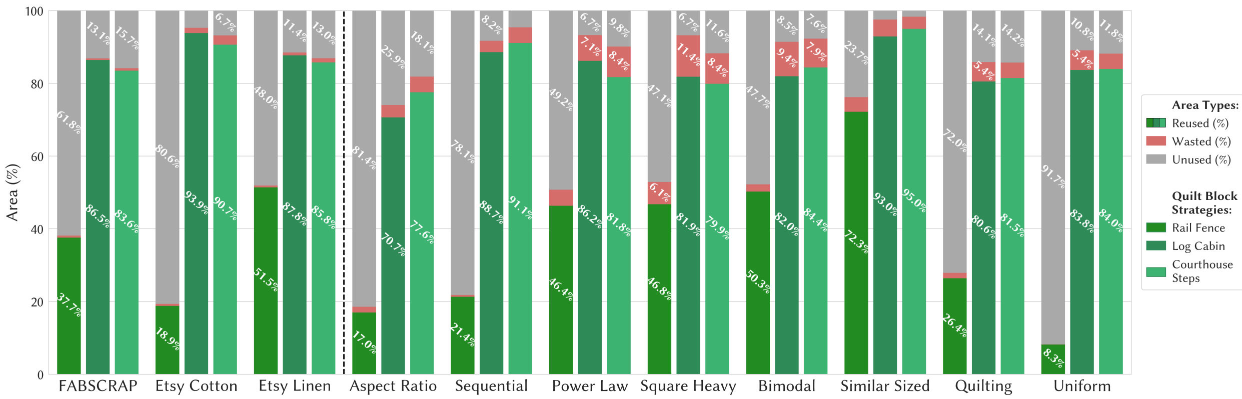


Figure 5.11: We plot the reused, wasted, and unused percentages of a given test case for each quilt-block-inspired packing strategy. The first three test cases are sets of actual fabric scraps and the other eight are generated test sets of different shape distributions. Aspect ratio: 70% are high-aspect-ratio rectangles (between 5:1 and 10:1); Bimodal: 60% near min size, 40% near max size; Power Law: 60% small sizes, 30% medium sizes, 10% large sizes; Sequential: uniformly increasing sizes; Similar Sized: the sizes are within 10% to 20% of each other; Square Heavy: 80% are squares; Quilting: 80% are standard quilting fabric sizes such as Jelly Rolls (2.5 in. strips), Charm Packs (5 in. squares), Cake Layers (10 in. squares), Coins (6.5 in. squares); Uniform: all random sizes.

We use the following metrics when evaluating the algorithm efficiency: (1) reused: the total area of scrap pieces that get reused in the patchwork design; (2) wasted: the total area of cut-off pieces that are too small to be reused again; (3) unused: scrap pieces that are not used in the current design but can be used in a different design. The breakdown of percentages of reused, wasted, and unused scraps for each test case using each quilt-block-inspired packing strategy is shown in Figure 5.11.

Since no user would be present for this evaluation, we make our method automatic by always selecting the least wasted area strip option to pack in each step until no more options are available.

On average, the wasted percentage is 3.63%, with the three fabric sets' average being 0.75% and the generated test sets' average being 4.71%. The Log Cabin and Courthouse Steps strategies range in efficiency 70-95% and Rail-Fence ranges 8-52%. The Rail Fence has a much lower reuse efficiency because it does not pack until after creating 12 strips (Figure 5.5).

We observe that among the generated sets, the set with similar-sized scraps appears to have the highest reuse efficiency using all three strategies. When there are imbalances between sizes of scraps, both Log Cabin and Courthouse Steps produce relatively higher waste. This could suggest that a collection of similar-sized scraps could get reused more efficiently through our packing method (comparing Similar Sized with Power Law and Bimodal in Figure 5.11). All three packing strategies produce relatively higher waste if 80% of the shapes are squares (Figure 5.11 Square Heavy). This means that to reuse scraps more effectively through ScrapPack, it would be good to have a more balanced ratio between rectangular pieces and square pieces.

(a) Our method with three packing strategies. The packing results' areas are as follows: Rail Fence - 343.55 in^2; Log Cabin - 789.02 in^2; Courthouse Steps - 762.17 in^2.

(b) Three automatic bin packing methods. The packing results' largest gap-free areas out of total bin areas are as follows: Strip Packing - 325.21 / 1051.49 in^2; Guillotine Packer - 323.13 / 968.83 in^2; RectPack - 485.41 / 946.00 in^2.

Figure 5.12: Packing results comparison between ScrapPack strategies and three bin packing methods. The dark red rectangles indicate the largest gap-free rectangles within the bin packing results.

5.6.2 Comparing with Automatic Bin Packing

The three open-source implementations we compared against were Strip Packing [131] (based on [215]), Guillotine Packer [160] (based on [99]), and RectPack [164] (based on [99, 85]).

Comparison Setup The following setup was employed to make the comparison between our methods and the automatic bin packing methods:

1. Because bin packing methods require a bin size to start with, we applied a grid search for the best width and height parameters so that all fabric scraps could be packed into a single bin. Our method does not require a bin size but the Rail Fence packing strategy requires a start length parameter (the width of the first sub-block) so we applied a grid search for this parameter in a similar fashion.
2. Our algorithm always generates hole-free packing. Since bin packing does not guarantee that the result has no holes, we compute the largest gap-free area within the packed bin (dark red rectangle in Figure 5.12b).
3. Since bin packing methods do not consider seam allowances, we set the seam allowance to zero in our method.
4. Since bin packing methods are automatic, similar to Section 5.6.1, we make our method automatic by always selecting the least wasted area strip option to pack in each step until no more options are available.

Figure 5.12 shows the packing results on the FABSCRAP scraps test set. We chose this test case because the gap-free area percentages of the bin packing methods were the highest (the other fabric scrap sets' results are in Appendix B).

For this test case, our proposed packing algorithm reuses fabric scraps efficiently, generating very little waste (<1%, see Figure 5.11 first three bars). Log Cabin and Courthouse Steps make use of 86.5% and 83.6% scraps, respectively. Rail Fence makes use of fewer scraps and produces a smaller patchwork because it stops after completing the 12-strip pattern.

Compared to our approach, where not all fabrics are used, bin packing methods can always use all of the fabrics when packing towards the right bin size. However, the resulting packing results have gaps and thus are not suitable to be used as a plan to make a sheet of fabric from scraps. If we look at the dark red rectangles showing the gap-free regions, their sizes are similar to that of the Rail Fence result and moderately smaller than that of the Log Cabin or Courthouse Steps result. In addition, despite this being a no-user comparison, the visual styles of the results of our method and the bin packing methods are distinct because our packing is guided by quilt blocks with specific strip structures and layouts.

5.7 User Evaluation

To understand how ScrapPack can support crafters and artists in efficiently reusing scraps while providing sufficient user control over the final patchwork design, we conducted a user evaluation with crafters.

5.7.1 Participants

We recruited 10 participants with varying levels of experience with textile crafts, using scraps, and quilting. As shown in Table 5.2, textile crafting experience ranged from a few months to 50 years, eight participants have used scraps in some projects before, and the number of quilts made ranged from none to 20. Among the 10 participants, 3 are members of a local quilt guild (P6, P7, P9), and the other 7 participants are part of local crafting interest groups.

Table 5.2: Summary of user study participants. Participants had been sewing for 1-50 years, and had made 0-20 quilts, reflecting a range of experience levels with patchwork. All but P2 and P3 had experience working with scrap fabrics.

ID	Gender	Age	Years	Used Scraps?	Amount of Scraps	# Quilts
P1	F	25–34	15	Yes	About same	2
P2	F	25–34	~2	No	Far fewer	0
P3	F	25–34	2	No	About same	0
P4	M	25–34	~5	Yes	A bit fewer	1
P5	NB	25–34	2	Yes	A bit fewer	0
P6	F	35–44	2	Yes	Far fewer	5
P7	F	25–34	5+	Yes	About same	10~15
P8	F	55–64	~50	Yes	Far fewer	0
P9	F	35–44	11	Yes	A bit more	20
P10	F	25–34	~1	Yes	A bit fewer	1

5.7.2 Study Procedure

The study sessions were conducted one-on-one, and each lasted 60-75 minutes. The sessions happened in a lab environment where participants came in and used the tool on the experimenter's laptop or over Zoom where participants used the tool through Zoom remote control.

The experimenter started by obtaining consent to record the participant's screen actions and the conversations during the session. In the first 15-25 minutes, the experimenter first demonstrated the tool with a test set of images. The participant then freely interacted with the tool and asked any questions during the process to familiarize with the tool. Next, the participant was asked to complete two design tasks.

Task 1: [Playing with Contrast] Imagine you're designing a pillowcase for a living room. Create a design that makes use of the contrast of colors.

Task 2: [Scrappy Fabric for Tote Bag] Imagine you are working with a stash of fabric scraps that you accumulated over the years. Now you want to create a new piece of fabric from these fabric scraps that can be used to make a tote bag (assuming that for a simple tote bag design, this new piece of fabric will be folded in half and sewn together on two sides, leaving one opening, to form the bag body, then the bag handles will be attached later).

Following each task, the experimenter asked open-ended questions specific to each task, focusing on how the participants approached the task and how they thought about the experience using the tool.

After the tasks, participants were asked to fill out a post-study questionnaire on the tool's usability, what they liked or not liked about the tool, and how it could be improved. The experimenter also asked some general questions about the tool and the participants' experiences with using scraps.

At the end of the sessions, the experimenter followed up with the participants asking if they would like to try to use ScrapPack with their own scraps. Four participants showed interest, and we were able to schedule in-person sessions with P4 and P5. Prior to the follow-up sessions, participants brought their scraps to the experimenter, who helped with digitizing the fabrics. During the follow-up sessions, the participants designed using their own scraps and fabricated their own designs (see Figure 5.15).

5.7.3 Results

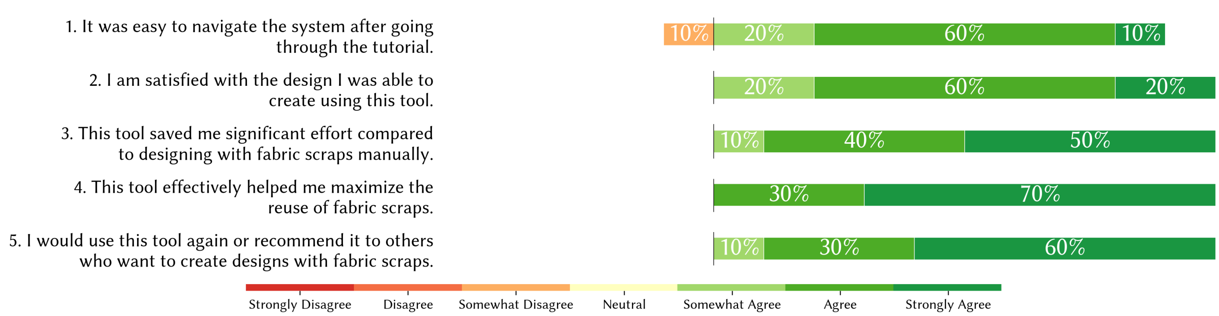


Figure 5.13: Summary of Likert-scale usability questions. Participants found ScrapPack supported them in maximizing scrap usage and allowing them to create satisfying designs. Most participants found the tool easy to navigate. P10, who found the tool the most difficult, wanted to swap out fabrics in the generated options, which required additional effort to modify the automatically suggested bins.

Based on the user evaluation results, participants generally felt positive about ScrapPack, finding it supporting them with maximizing scrap usage and allowing them to create satisfying designs (Figure 5.13).

Tradeoffs between scrap reuse efficiency and the visual of the design

Participants generally agree that ScrapPack effectively helped them maximize scrap fabric usage (Figure 5.13 No.4). But interestingly, when asked about whether they prioritized saving materials during the design tasks, most of them said they did not consider the wasted area of each strip option and the reuse efficiency of the current design, and for different reasons.

Several participants felt the tool did a good job of keeping track of the remaining scraps and the scrap reuse efficiency for them. P6 noted, "I did not think so much about the scrap utilization ... I knew I could see that it was keeping track of it." P8 said, "This puts a number to [reuse efficiency], which is kind of nice ... I can focus on the creativity and what my eye sees." Similarly, P10 highlighted that the tool "eases the cognitive load of trying to save the material." Some participants (P4, P9) did not focus as much on efficiency and said they were just happy to be using any scraps from their pile.

Participants also appreciated that the tool would save any cut-off portions that are still large enough for later use. P1 said "I didn't really think about the scrap material saving. I think the fact that there are so many scraps made me feel like ... I could just pick and choose anything I wanted ... I like the idea that when you cut off the scraps, you're going to have more." She found the experience "empowering" because it changed her mindset from thinking how she could "simply utilize all the scraps as much as possible" to thinking how she could actually design something she wants with the scraps. P2 thought the tool helped "lower the mental load of keeping track of all of the different scraps and what they might look like when halfway used." P5 also really liked that "the tool was actually keeping track of that for me by virtually cutting it and reincorporating it." P8 noted though that aiming to reuse the leftovers put additional pressure on cutting the pieces correctly.

Supporting design exploration and providing user control

When being asked about what they liked most about the tool, participants highlighted the automatic binning (8 of 10) and strip option generation (7 of 10) features. They valued the abundance of variations and the ability to rearrange, sort, and personalize options.

Participants said that they liked the variation in the design suggestions. For example, P2 said "I like that it inspired me on how different bins could look or what was missing" when talking about the automatic binning. P2, P9, and P10 liked the novelty of seeing subsequent generated strip options. P2 reflected in the post-study conversations that during Task 2 (see Figure 5.14 left), she was at one point not sure what would be good to add to the design next, but when she scrolled through the available options and saw a strip option with yellows, she found it effectively counterbalanced the blues and immediately knew that it was what she wanted. P9 had a similar experience: after Task 1 (see Figure 5.14 right), she said "I liked ... whatever came up ... that little tiny stripe of blue plaid is kind of delightful there." And she found the abundant options available helpful in creating the chaotic look she wanted in her design. P10 found the Wasted Area ranking

Figure 5.14: Highlighting two participant designs. P2's Task 2 design: the left yellow-ish strip (in red rectangle) surprised P2 and she liked how it counterbalanced the blues well and without seeing it, she thought she would not think yellow was what she wanted. P9's Task 1 design: the strip towards the bottom (in red rectangle) had a small piece of blue plaid which P9 found unexpected and liked how it looked.

(Ranking Criterion 1) especially helpful: "The available options ... actually could go together well in ways that I did not consider ... Sort[ing] by wasted area pushed me to try combinations that I'd not do otherwise."

Overall, the strip-level generation offered both structure and design freedom, as can be seen in the variety of designs created by participants during the user study (Appendix A). Participants generally felt they had enough agency to intervene and modify generated strips. They enjoyed the ability to rearrange fabrics within a strip option, though several requested even more flexibility (see Section 5.7.3).

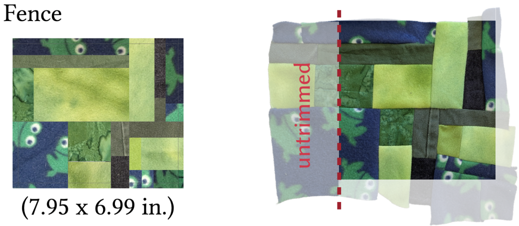
The value of digitally laying out scraps

Participants consistently found that designing digitally with the tool saved time and effort (Figure 5.13 No.3). P2 liked "being able to plan out the entire patchwork before getting started," and P9 thought "being able to visualize the end product without cutting into my fabric and making a mistake is a really nice feature." P4 echoed that this tool would help him avoid mistakes: "if I was doing this by myself there would be a lot more times where I'd have to pull out the seam and realize things don't match as much as I thought it would, or the strip is just a little too short because I didn't consider the seam allowances."

The alternative of designing this type of scrap sheet manually was considered overwhelming and time-consuming. P3 said, "If I were to [design manually], it would be very difficult ... I'd probably have to do it on the floor, too." P4 echoed that the tool would save "hours and hours of just like sitting on the floor shuffling things around." P8 also thought that "I would get really overwhelmed with all the fabrics just on my own."

Fabrication instructions

All participants found the instructions helpful and felt confident that they could put together their design with the instructions. P2 said "I really liked how it went into useable instructions." P4 agreed that "The explicit instructions are really helpful." P4 and P5 both successfully constructed their designs from the tool's outputs during follow-up sessions. Interestingly, participants felt empowered to personalize or improvise with the instructions. For example, P4 swapped in a larger piece of frog fabric to enlarge the final result, illustrating how the instructions could serve as a good starting point rather than a rigid plan.



(a) Since P4 did not trim the fabricated result, we did not include it in Table 5.1. P4 also decided to swap in a larger piece of the frog fabrics in the last strip (there were many frog fabric scraps) so that he did not need to trim off the design and could get the final result to be bigger. P4 intended to use this for his dog's jacket.

(b) P5 followed the instructions closely, except in the last round on the leftmost vertical strip, where they reordered the fabrics to avoid two similar-looking scraps being next to each other. P5 intended to make this into a wall hanging or a bag for their portable loom.

Figure 5.15: Designs created by P4 and P5 with their own scrap fabrics and fabricated.

Project ideas with the tool

Participants envisioned a wide range of potential uses for patchworks created with the tool, including household items (pot holders, bags, cases, book covers, blankets, napkins, tablecloths, pillow cases, tea cozies), clothing items (quilt jackets, hats), and quilts (lap quilts, scrappy quilts, baby quilts). P4 said his friends would love to use this tool to design scrappy jackets. P8 felt motivated to start working on a scrappy quilt. P10 was inspired by the designs she created with the tool and would like to upcycle her wardrobe through patchwork designs.

Feature suggestions

Participants had various suggestions for improving the tool, mainly centered around better visualization and more flexible user control. These include (1) enhancing binning functionalities (allowing meta-bins for cross-bin strip option generation, automatic sorting into equal-sized bins); (2) easier strip option filtering (starring options or fabrics, specifying strip thickness ranges based on prior strips, contrasting the strip with the current design instead of within the strip); (3) more flexible customization of generated strip options (rotating directional fabrics, swapping individual fabrics in a strip option); (4) better conveying the sense of scale of the design and the strip options through improved dimension visualization and previewing the selected strip; (5) incorporating design guides such as bin templates for specific packing strategies (e.g., automatically alternate between bins for the three-strip groups in Rail Fence), sewing pattern templates for common upcycling projects to guide the design process (e.g., overlaying a tote bag pattern on the canvas and visualizing the 3D shape), and the capability to create multiple patchworks in a multi-block layout. These feature requests offer valuable directions for future tool development.

5.8 Discussion and Future Work

As discussed in Section 5.7.3, participants found ScrapPack helped them create satisfying patchwork designs using scraps efficiently, and liked the variety of options generated. However, the system has several limitations and opportunities for future work.

Digitizing fabric scraps Although we see the value of digitizing fabric scraps and designing digitally from Section 5.7.3, the current semi-automatic method for fabric image extraction (Section 5.4.1) could feel like a burden for a user if they keep a large number of fabric scraps. P7 suggested that it could be nice to allow users to describe what fabrics they have available and generate some palette matching the description for getting an initial idea. P4 suggested making it a mobile app that a user could use to keep track of all the scraps produced every time they work on a new sewing/crafting project. Future work could try to improve the current extraction method to be fully automatic and also incorporate the ideas suggested by the participants.

Handling irregular scraps and packing targets We only considered packing rectangular fabric scraps in this work (although photos of scrap pieces can be non-rectangular, we extracted rectangles from them). This simplification can lead to a small amount of error in our efficiency calculation, depending on these input shapes. However, scraps from some kinds of projects, such as garment making (as mentioned by P1 and P3), could have very irregular or curved shapes.

Irregular shape packing, like 2D bin packing, is also an extensively studied problem and has existing methods. Considering more irregular shapes and curved edges would be a challenging and interesting future direction. We could start with incorporating non-rectangular shapes like triangles, since triangles are also very common in quilt blocks. This would allow the tool to support more quilt blocks, providing more visual variety in patchwork designs.

We could also consider packing toward irregular shapes. For example, many clothing panels, such as a top with armholes, have non-rectangular shapes. Being able to pack towards a non-rectangular target shape could allow users to place their scraps with respect to the placement on the final garment, rather than packing a sheet, which they have to cut later.

Balancing between design freedom and scrap reuse efficiency Our technical evaluation (Section 5.6) and user study results (Section 5.7) suggest that breaking down patchwork design into strips is an effective way to allow both user control and material waste minimization. In addition, the tool can already support users in creating a diverse set of designs with the three quilt blocks, and Log Cabin and Courthouse Steps also achieve relatively high reuse efficiency (>80% in typical cases).

To further improve the design variations while maintaining material efficiency, it would be useful to understand what makes a quilt block (i.e., a 2D space partitioning) suitable for achieving a high scrap reuse efficiency. The current three blocks supported are categorized as *strip quilt blocks* [18], and we could investigate how other blocks in this family would perform, including Roman Square, Roman Stripe, and variations of Rail Fence (4 or 5 strips in a sub-block).

Domain-aware packing Although this work focuses on fabric patchwork as an application of the quilt-block-inspired packing algorithm, we believe that the same idea applies in other domains that care about both visual aesthetics and material efficiency. For example, applications like leather working, ceramic tile installation, and upholstery covering could benefit from the strip-based design approach, while potentially having other considerations specific to the domain such as minimizing the lengths of cuts. It would be interesting to see if there are more suitable intermediate design abstractions for a different domain as well.

Chapter 6

Conclusion

This dissertation contributes a way to look at sustainability through the lens of design and fabrication. It focuses on using proper abstractions of design spaces to build systems that support exploration, evaluation, and optimization of more sustainable designs. We demonstrate this approach across domains such as electronics, home accessories, and textile products. In each case, we enabled sustainable practices like recycling, reuse, and waste reduction. By developing suitable representations, we made these design tools more accessible to everyday users, helping them navigate constraints from materials and fabrication processes while still creating designs that are more sustainable in some ways.

6.1 Future Work

The work we discussed in this dissertation presents several directions for future work.

6.1.1 More methodological way for creating domain abstractions

The abstractions discussed in this dissertation were designed through either formative studies or ad-hoc research. We speak to experts through interviews, or consult books and existing literature in a specific domain, to come up with insights that help guide us in finding the right abstractions. Ideally, there should be a more methodical way to find a good abstraction, and it would be interesting to understand what are the traits of such abstractions. One direction is to consider library learning, which is a concept from the field of programming languages. Given a programming language (usually a domain-specific language) and a set of programs in that language, we can find a library of common functionalities. In the context of fabrication, the idea of "fabrication is program" has shown great potential in many domains [135, 202, 98, 28, 128, 221]. Thus, it is a promising direction to try to represent designs, or the fabrication instructions of the designs, as programs, and learn from them a library of functions that can more compactly represent the design space that is fabricable. If we can evaluate the sustainability metrics associated with the library functions, this could further allow us to incorporate sustainability into program generation and optimization.

6.1.2 Co-optimizing fabrication plan and design

Switching to clean energy sources and improving energy efficiency are still the most effective strategies for carbon emission reduction. It would be helpful to understand from a design perspective, how a design relates to the energy consumption of its fabrication process. This could mean coming up with models for the machines and mapping the machine actions with fabrication instructions for obtaining rough estimates. As such, there would be oppportunities for fabrication plan optimization. We could either decouple this from the design process completely through a bilevel optimization, or employ a turn-taking scheme that alternates between optimizing the design and optimizing the fabrication plan.

6.1.3 Choosing the sustainability metrics

As introduced in Chapter 3, LCA is a standard but time-consuming way to evaluate environmental impact in the manufacturing industry. In order to incorporate sustainability feedback into the early stage of the design process, we need to seek alternatives that can still meaningfully provide feedback on how sustainable the design might be. Chapter 3 circumvents the issue by using comparative assessment instead, which is a great strategy to employ when absolute quantities are not necessary, while in Chapter 5, since our focus is specifically on scrap fabrics, there happens to be a straightforward way to equivalently represent the environmental impact (through the wasted area). For future work, it would be interesting to come up with other creative ways to measure environmental impacts that are related to the domain-specific representations so that the EIs can be efficiently evaluated.

The choice of the sustainability metrics impacts how to navigate the design space with tradeoffs between sustainability and other performance metrics. Adding sustainability axes to the performance space, without doubt, makes the space higher-dimensional and therefore harder to explore. If the sustainability metric we choose can be a surrogate from another performance metric that is already being evaluated, it will reduce the complexity of the problem. In addition, it would also be interesting to see how much of the design exploration can be automated, either through optimization and search methods, or schemes like reinforcement learning with reward functions defined through the performance metrics.

6.1.4 Underexplored sustainable design principles

Looking at rubric items outlined by Blevis [12] for reasoning about sustainable interaction design from the perspective of sustainability, this dissertation mainly touched on *disposal* (as in causing waste during the manufacturing processes), *salvage* (as in recovering waste), *recycling*, *remanufacturing for reuse*, *reuse as is*, and *finding wholesome alternatives to use*. We now discuss the remaining rubric items that this dissertation did not explore and their corresponding future directions.

Circular Economy

This topic distantly relates to *sharing for maximal use* (as in sharing the product due to using shared resources) from SID [12]. The idea of circular economy, which in the ideal case, will not generate waste and everything is used to remanufacture, has been explored in many contexts [24, 168, 103]. This is often considered on the scale of a community, not just an individual. From a computational design perspective, circular economy calls for a design system that considers a set of designs. We need to take into account how the other designs will impact one design because they are now interdependent. This presents a significantly larger design space and more complex constraints, and is an exciting direction to explore further.

Mend, Repair, and Renew

This topic corresponds to *achieving longevity of use*, and is slightly related to *achieving heirloom status* (as in the item itself having long-lived appeal that motivates preservation) and *active repair of misuse* (as in repairing harmful effects by substituting out the unsustainable uses) from SID [12]. Although we did not mention repair when introducing the methodology of LCA (Section 3.1), it is an important scenario that most LCAs with an end-of-life analysis will consider. In the context of LCA, we usually assume a certain lifespan for the product, and the number of repairs a user might do; these information directly impact the EI of the Use phase.

In general, making the lifespan of the product longer also means lower demand for new products of the same functionality, which is more sustainable. Repairing is common for many products because it is usually much more cost effective than building a completely new one. For example, for textiles and home accessories, there is a long history of mending and darning. Some techniques even turned into artistic practices, such as sashiko stitching (reinforcing fabric patches, usually old clothing), and jinshan or kintsukuroi (repairing items with golden lacquer). This is also a topic of interest in the fabrication research community; prior research has been done on how to enable repairing and renewing (Section 2.3.2).

Incorporating repairability into designs could mean that the design is modularized so that we can easily swap out faulty modules, or more generally, the design is easy to disassemble (or unmake [173]) and thus easy to be repaired. These considerations can be formalized as paradigms to construct a design and therefore the design system. In addition, we can derive sustainability metrics other than environmental impacts, including the time it takes to disassemble a design, the number of uses a design could sustain, and the cost of obtaining one of the most easily damaged modules. We can build design systems that take these metrics into account to enable designs that are easy to repair, and thus more sustainable.

6.1.5 Sustainability beyond design systems for sustainable fabrication

As Ceschin and Gaziulusoy [30] pointed out, in order to actually drive sustainability changes in the society, we need to shift from insular, individual technological innovations to more systemic, societal approaches. In addition, we did not deploy any of the work discussed in this dissertation for the general public to use.

More evaluations of these technological advances in design systems, including how do people perceive the designs created using these tools, and whether using such tools affects other aspects in their lifestyle, could be informative for future design system developement. Truly understanding the impact of the technological innovations on everyday users would go a long way for fostering awareness and understanding what might actually drive behavioral change in our society.

Bibliography

[1] Bilge Acun, Benjamin Lee, Fiodar Kazhamiaka, Kiwan Maeng, Udit Gupta, Manoj Chakkaravarthy, David Brooks, and Carole-Jean Wu. 2023. Carbon Explorer: A Holistic Framework for Designing Carbon Aware Datacenters. In *Proceedings of the 28th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2 (ASPLOS 2023)*. Association for Computing Machinery, New York, NY, USA, 118–132. https://doi.org/10.1145/3575693.3575754

[2] Ali Asghar Adibi. 2021. A Brief History of Collage. In *Collage: A Process in Architectural Design*, Ali Asghar Adibi (Ed.). Springer International Publishing, Cham, 1–5. https://doi.org/10.1007/978-3-030-63795-8_1

[3] Amazon. 2023. Amazon Devices Product Carbon Footprint Methodology. https://sustainability.aboutamazon.com/devices-carbon-methodology.pdf

[4] Abul Al Arabi and Jeeeun Kim. 2022. Augmenting Everyday Objects into Personal Robotic Devices. In *SIGGRAPH Asia 2022 Emerging Technologies*. ACM, Daegu Republic of Korea, 1–2. https://doi.org/10.1145/3550471.3564763

[5] Abul Al Arabi, Jiahao Li, Xiang 'Anthony Chen, and Jeeeun Kim. 2022. Mobiot: Augmenting Everyday Objects into Moving IoT Devices Using 3D Printed Attachments Generated by Demonstration. In *CHI Conference on Human Factors in Computing Systems*. ACM, New Orleans LA USA, 1–14. https://doi.org/10.1145/3491102.3517645

[6] Vicente Arroyos, Maria L K Viitaniemi, Nicholas Keehn, Vaidehi Oruganti, Winston Saunders, Karin Strauss, Vikram Iyer, and Bichlien H Nguyen. 2022. A Tale of Two Mice: Sustainable Electronics Design and Prototyping. In *Extended Abstracts of the 2022 CHI Conference on Human Factors in Computing Systems (CHI EA '22)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3491101.3519823

[7] Omri Azencot, Etienne Corman, Mirela Ben-Chen, and Maks Ovsjanikov. 2017. Consistent functional cross field design for mesh quadrangulation. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–13.

[8] Alba Bala, Marco Raugei, Gabriela Benveniste, Cristina Gazulla, and Pere Fullana-i Palmer. 2010. Simplified tools for global warming potential evaluation: when 'good enough' is best. *The International Journal of Life Cycle Assessment* 15, 5 (June 2010), 489–498. https://doi.org/10.1007/s11367-010-0153-x

[9] Sebastiano Battiato, Gianluigi Ciocca, Francesca Gasparini, Giovanni Puglisi, and Raimondo Schettini. 2007. Smart Photo Sticking. In *Adaptive Multimedial Retrieval: Retrieval, User, and Semantics: 5th International Workshop, AMR 2007, Paris, France, July 5-6, 2007 Revised Selected Papers*. Springer-Verlag, Berlin, Heidelberg, 211–223. https://doi.org/10.1007/978-3-540-79860-6_17

[10] Hugh Beyer and Karen Holtzblatt. 1999. Contextual design. *interactions* 6, 1 (1999), 32–42.

[11] Gurbakhash Singh Bhander, Michael Hauschild, and Tim McAloone. 2003. Implementing life cycle assessment in product development. *Environmental Progress* 22, 4 (2003), 255–267. https://doi.org/10.1002/ep.670220414 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/ep.670220414

[12] Eli Blevis. 2007. Sustainable interaction design: invention & disposal, renewal & reuse. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '07)*. Association for Computing Machinery, New York, NY, USA, 503–512. https://doi.org/10.1145/1240624.1240705

[13] David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum* 30(2) (2011), 375–384. https://doi.org/10.1111/j.1467-8659.2011.01868.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.01868.x

[14] David Bommes, Timm Lempfer, and Leif Kobbelt. 2011. Global Structure Optimization of Quadrilateral Meshes. *Computer Graphics Forum* 30, 2 (April 2011), 375–384. https://doi.org/10.1111/j.1467-8659.2011.01868.x

[15] David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. 2013. Quad-Mesh Generation and Processing: A Survey: Quad-Mesh Generation and Processing. *Computer Graphics Forum* 32, 6 (Sept. 2013), 51–76. https://doi.org/10.1111/cgf.12014

[16] David Bommes, Henrik Zimmer, and Leif Kobbelt. 2009. Mixed-integer quadrangulation. *ACM Transactions on Graphics* 28, 3 (July 2009), 1. https://doi.org/10.1145/1531326.1531383

[17] Kim Brackett. 2013. *Scrap-basket beauties: quilting with scraps, strips, and jelly rolls*. Martingale, Bothell, WA. OCLC: 807045408.

[18] Barbara Brackman. 2021. *Encyclopedia of Pieced Quilt Patterns* (3rd ed.). The Electric Quilt Company, Bowling Green, OH. https://electricquilt.com/online-shop/encyclopedia-of-pieced-quilt-patterns/

[19] JC Briar. 2019. Stitch Maps. https://stitch-maps.com/

[20] Ole Broberg and Per Christensen. 1999. LCA experiences in Danish industry: results of a survey. *The International Journal of Life Cycle Assessment* 4 (1999), 257–262.

[21] Ann Budd. 2002. *The knitter's handy book of patterns: basic designs in multiple sizes & gauges*. Interweave Press, Loveland, CO.

[22] David V. Campbell. 2010. Yield modeling of 3D integrated wafer scale assemblies. In *2010 Proceedings 60th Electronic Components and Technology Conference (ECTC)*. IEEE, Las Vegas, NV, USA, 1935–1938. https://doi.org/10.1109/ECTC.2010.5490688

[23] Marcel Campen and Leif Kobbelt. 2014. Dual strip weaving: interactive design of quad layouts using elastica strips. *ACM Transactions on Graphics* 33, 6 (Nov. 2014), 1–10. https://doi.org/10.1145/2661229.2661236

[24] Mengyu Cao. 2017. Lunhui: Upcycled Creations. In *Proceedings of the 2017 ACM SIGCHI Conference on Creativity and Cognition (C&C '17)*. Association for Computing Machinery, New York, NY, USA, 269–271. https://doi.org/10.1145/3059454.3078710

[25] Christopher Carlson, Nina Paley, and Theodore Gray. 2015. Algorithmic Quilting. In *Proceedings of Bridges 2015: Mathematics, Music, Art, Architecture, Culture*. Bridges, Baltimore, Maryland, USA, 231–238. https://archive.bridgesmathart.org/2015/bridges2015-231.html#gsc.tab=0 ISSN: 1099-6702.

[26] Melanie Carrico and Victoria Kim. 2014. Expanding zero-waste design practices: a discussion paper. *International Journal of Fashion Design, Technology and Education* 7, 1 (Jan. 2014), 58–64. https://doi.org/10.1080/17543266.2013.837967 Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/17543266.2013.837967.

[27] Mary Carson. 2023. Recycling in textile art: Five artists – TextileArtist. https://www.textileartist.org/recycling-in-textile-art/ Section: Discover....

[28] Dan Cascaval, Rastislav Bodik, and Adriana Schulz. 2023. A Lineage-Based Referencing DSL for Computer-Aided Design. *Proc. ACM Program. Lang.* 7, PLDI (June 2023), 109:76–109:99. https://doi.org/10.1145/3591223

[29] Marilyn Caylor. 2019. 75 super easy ways to organize your entire home. https://homehacks.co/easy-home-organizational-tips/

[30] Fabrizio Ceschin and Idil Gaziulusoy. 2016. Evolution of design for sustainability: From product design to design for system innovations and transitions. *Design Studies* 47 (Nov. 2016), 118–163. https://doi.org/10.1016/j.destud.2016.09.002

[31] Tejaswini Chatty, Bryton L. Moeller, Ioana A. Pantelimon, Catherine D. Parnell, Tahsin M. Khan, Lise Laurin, Jeremy Faludi, and Elizabeth L. Murnane. 2024. EcoSketch: Promoting Sustainable Design through Iterative Environmental Assessment during Early-Stage Product Development. *ACM J. Comput. Sustain. Soc.* 2, 2 (June 2024), 22:1–22:29. https://doi.org/10.1145/3648436

[32] Xiang 'Anthony' Chen, Stelian Coros, and Scott E. Hudson. 2018. Medley: A Library of Embeddables to Explore Rich Material Properties for 3D Printed Objects. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Montreal QC Canada, 1–12. https://doi.org/10.1145/3173574.3173736

[33] Xiang 'Anthony' Chen, Stelian Coros, Jennifer Mankoff, and Scott E. Hudson. 2015. Encore: 3D Printed Augmentation of Everyday Objects with Printed-Over, Affixed and Interlocked Attachments. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 73–82. https://doi.org/10.1145/2807442.2807498 event-place: Charlotte, NC, USA.

[34] Xiang 'Anthony' Chen, Jeeeun Kim, Jennifer Mankoff, Tovi Grossman, Stelian Coros, and Scott E. Hudson. 2016. Reprise: A Design Tool for Specifying, Generating, and Customizing 3D Printable Adaptations on Everyday Objects. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology (UIST '16)*. ACM, New York, NY, USA, 29–39. https://doi.org/10.1145/2984511.2984512 event-place: Tokyo, Japan.

[35] Kyung Yun Choi and Hiroshi Ishii. 2021. Therms-Up!: DIY Inflatables and Interactive Materials by Upcycling Wasted Thermoplastic Bags. In *Proceedings of the Fifteenth International Conference on Tangible, Embedded, and Embodied Interaction*. ACM, Salzburg Austria, 1–8. https://doi.org/10.1145/3430524.3442457

[36] Emily Cier. 2011. *Scrap republic: 8 quilt projects for those who love color.* C & T Pub, Lafayette, CA. OCLC: 699003159.

[37] Marge M Coahran and Eugene Fiume. 2005. Sketch-Based Design for Bargello Quilts. *SBM* 165 (2005), 174.

[38] European Commission. 2017. Guidance for the development of Product Environmental Footprint Category Rules (PEFCRs). https://eplca.jrc.ec.europa.eu/permalink/PEFCR_guidance_v6.3-2.pdf

[39] European Commission. 2022. Environmental Footprint reference package 3.1. https://eplca.jrc.ec.europa.eu/LCDN/developerEF.html

[40] 5-Minute Crafts. 2022. 5-Minute Crafts — Learn. Create. Improve. https://5minutecrafts.site/

[41] Keenan Crane, Mathieu Desbrun, and Peter Schröder. 2010. Trivial Connections on Discrete Surfaces. *Computer Graphics Forum* 29, 5 (Sept. 2010), 1525–1533. https://doi.org/10.1111/j.1467-8659.2010.01761.x

[42] U-M CSS. 2024. Municipal Solid Waste Factsheet. https://css.umich.edu/publications/factsheets/material-resources/municipal-solid-waste-factsheet

[43] U-M CSS. 2024. U.S. Material Use Factsheet. https://css.umich.edu/publications/factsheets/material-resources/us-material-use-factsheet

[44] Zero Waste Daniel. 2024. Upcycled Clothing and Accessories. https://zerowastedaniel.com/

[45] Scott Davidoff, Nicolas Villar, Alex S. Taylor, and Shahram Izadi. 2011. Mechanical hijacking: how robots can accelerate UbiComp deployments. In *Proceedings of the 13th international conference on Ubiquitous computing*. ACM, Beijing China, 267–270. https://doi.org/10.1145/2030112.2030148

[46] Leonardo de Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, C. R. Ramakrishnan and Jakob Rehof (Eds.). Springer Berlin Heidelberg, Berlin, Heidelberg, 337–340.

[47] Dell Inc. 2023. Understanding Life Cycle Assessments (LCAs), Product Carbon Footprints (PCFs), and the uses and limitations of PAIA, a streamlined LCA methodology. https://www.delltechnologies.com/asset/en-us/products/multi-product/industry-market/pcf-lca-whitepaper.pdf

[48] Dell Inc. 2024. Product Carbon Footprint (PCF) | Dell USA. https://www.dell.com/en-us/dt/corporate/social-impact/advancing-sustainability/climate-action/product-carbon-footprints.htm

[49] P Dempsey. 2017. The Teardown: iPad 2017 tablet computer. *Engineering & Technology* 12, 5 (2017), 80–81.

[50] Kristin N. Dew and Daniela K. Rosner. 2019. Designing with Waste: A Situated Inquiry into the Material Excess of Making. In *Proceedings of the 2019 on Designing Interactive Systems Conference (DIS '19)*. Association for Computing Machinery, New York, NY, USA, 1307–1319. https://doi.org/10.1145/3322276.3322320

[51] Olga Diamanti, Amir Vaxman, Daniele Panozzo, and Olga Sorkine-Hornung. 2015. Integrable polyvector fields. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–12.

[52] Jérémie Dumas, Jean Hergel, and Sylvain Lefebvre. 2014. Bridging the gap: automated steady scaffoldings for 3D printing. *ACM Transactions on Graphics* 33, 4 (July 2014), 98:1–98:10. https://doi.org/10.1145/2601097.2601153

[53] Katie Ebben. 2008. *Fabric scrapping*. Sterling, New York. OCLC: 213301313.

[54] Hans-Christian Ebke, David Bommes, Marcel Campen, and Leif Kobbelt. 2013. QEx: robust quad mesh extraction. *ACM Transactions on Graphics* 32, 6 (Nov. 2013), 1–10. https://doi.org/10.1145/2508363.2508372

[55] Econation. 2024. Sustainable Design Basics | Design for Sustainability basics. https://econation.one/sustainable-design-basics/

[56] Mariam Elgamal, Doug Carmean, Elnaz Ansari, Okay Zed, Ramesh Peri, Srilatha Manne, Udit Gupta, Gu-Yeon Wei, David Brooks, Gage Hills, and Carole-Jean Wu. 2023. Carbon-Efficient Design Optimization for Computing Systems. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems (HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3604930.3605712

[57] Nesma ElShishtawy, Pammi Sinha, and Julia A. Bennell. 2022. A comparative review of zero-waste fashion design thinking and operational research on cutting and packing optimisation. *International Journal of Fashion Design, Technology and Education* 15, 2 (May 2022), 187–199. https://doi.org/10.1080/17543266.2021.1990416 Publisher: Taylor & Francis _eprint: https://doi.org/10.1080/17543266.2021.1990416.

[58] Zachary Englhardt, Felix Hähnlein, Yuxuan Mei, Tong Lin, Connor Masahiro Sun, Zhihan Zhang, Shwetak Patel, Adriana Schulz, and Vikram Iyer. 2025. Incorporating Sustainability in Electronics Design: Obstacles and Opportunities. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3706598.3713299

[59] US EPA. 2025. Industry Sector Emissions. https://www.epa.gov/ghgemissions/industry-sector-emissions

[60] US EPA. 2025. Sources of Greenhouse Gas Emissions. https://www.epa.gov/ghgemissions/sources-greenhouse-gas-emissions

[61] European Commission. Joint Research Centre. 2023. *GHG emissions of all world countries: 2023.* Publications Office, LU. https://data.europa.eu/doi/10.2760/953322

[62] FABCYCLE. 2025. FABCYCLE | Giving Fabric Waste a Second Life – Deadstock & Free Scraps. https://www.fabcycle.shop/

[63] FABSCRAP. 2024. FABSCRAP. https://fabscrap.org

[64] Fairphone. 2024. Repairability and recyclers information on Fairphone 4. https://www.fairphone.com/wp-content/uploads/2022/09/FP4_Information-for-repairers-and-recyclers.pdf

[65] R. Featherstone. 1983. The Calculation of Robot Dynamics Using Articulated-Body Inertias. *The International Journal of Robotics Research* 2, 1 (March 1983), 13–30. https://doi.org/10.1177/027836498300200102

[66] Jasmin Fisher, Nir Piterman, and Rastislav Bodik. 2014. Toward Synthesizing Executable Models in Biology. *Frontiers in Bioengineering and Biotechnology* 2 (2014), 1–8. https://doi.org/10.3389/fbioe.2014.00075

[67] Fiyaa. 2013. 15 Cord Management Life Hacks for No More Tangled Wires. https://www.amazinginteriordesign.com/15-cord-management-life-hacks-for-no-more-tangled-wires/

[68] Hüseyin Fırat and Nuh Alpaslan. 2020. An effective approach to the two-dimensional rectangular packing problem in the manufacturing industry. *Computers & Industrial Engineering* 148 (Oct. 2020), 106687. https://doi.org/10.1016/j.cie.2020.106687

[69] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. 2009. IWIRES: An Analyze-and-Edit Approach to Shape Manipulation. *ACM Trans. Graph.* 28, 3, Article 33 (July 2009), 10 pages. https://doi.org/10.1145/1531326.1531339

[70] Stas Goferman, Ayellet Tal, and Lihi Zelnik-Manor. 2010. Puzzle-like Collage. *Computer Graphics Forum* 29, 2 (2010), 459–468. https://doi.org/10.1111/j.1467-8659.2009.01615.x

[71] Google. 2023. OR-tools. https://developers.google.com/optimization

[72] Katja Tasala Gradin and Anna Björklund. 2021. The common understanding of simplification approaches in published LCA studies—a review and mapping. *The International Journal of Life Cycle Assessment* 26, 1 (Jan. 2021), 50–63. https://doi.org/10.1007/s11367-020-01843-4

[73] Anhong Guo, Jeeeun Kim, Xiang 'Anthony' Chen, Tom Yeh, Scott E. Hudson, Jennifer Mankoff, and Jeffrey P. Bigham. 2017. Facade: Auto-generating Tactile Interfaces to Appliances. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*. ACM, Denver Colorado USA, 5826–5838. https://doi.org/10.1145/3025453.3025845

[74] Udit Gupta, Mariam Elgamal, Gage Hills, Gu-Yeon Wei, Hsien-Hsin S. Lee, David Brooks, and Carole-Jean Wu. 2022. ACT: designing sustainable computer systems with an architectural carbon modeling tool. In *Proceedings of the 49th Annual International Symposium on Computer Architecture (ISCA '22)*. Association for Computing Machinery, New York, NY, USA, 784–799. https://doi.org/10.1145/3470496.3527408

[75] Gurobi Optimization, LLC. 2024. Gurobi Optimizer Reference Manual. https://www.gurobi.com

[76] Kathrin Hanauer, Martin P. Seybold, and Julian Unterweger. 2023. Covering Rectilinear Polygons with Area-Weighted Rectangles. https://doi.org/10.48550/arXiv.2312.08540 arXiv:2312.08540 [cs].

[77] Sallieann Harrison. 2023. *Elevate your scrap sewing projects: 20+ beautiful techniques using your fabric stash*. Landauer Publishing, Mount Joy, PA. OCLC: 1414180422.

[78] Muhammad Hegazy, Kensuke Yasufuku, and Hirokazu Abe. 2021. Multi-objective optimization objectives for building envelopes: a review study. In *Imaginable Futures: Design Thinking, and the Scientific Method.* Architectural Science Association (ANZAScA), Online, New Zealand, 775–784. https://api.semanticscholar.org/CorpusID:259106859

[79] Birgitta Helmersson and Emli Bendixen. 2023. *Zero waste patterns: 20 projects to sew your own wardrobe*. Quadrille Publishing, London. OCLC: 1373396709.

[80] Chris Hendrickson, Arpad Horvath, Satish Joshi, and Lester Lave. 1998. Economic Input–Output Models for Environmental Life-Cycle Assessment. *Environmental Science & Technology* 32, 7 (April 1998), 184A–191A. https://doi.org/10.1021/es983471i Publisher: American Chemical Society.

[81] Megan Hofmann, Lea Albaugh, Ticha Sethapakadi, Jessica Hodgins, Scott E. Hudson, James McCann, and Jennifer Mankoff. 2019. KnitPicking Textures: Programming and Modifying Complex Knitted Textures for Machine and Hand Knitting. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology - UIST '19*. ACM Press, New Orleans, LA, USA, 5–16. https://doi.org/10.1145/3332165.3347886

[82] Megan Hofmann, Gabriella Hann, Scott E. Hudson, and Jennifer Mankoff. 2018. Greater than the Sum of its PARTs: Expressing and Reusing Design Intent in 3D Models. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. ACM, Montreal QC Canada, 1–12. https://doi.org/10.1145/3173574.3173875

[83] Alexander Hollberg, Gianluca Genova, and Guillaume Habert. 2020. Evaluation of BIM-based LCA results for building design. *Automation in Construction* 109 (2020), 102972. https://doi.org/10.1016/j.autcon.2019.102972

[84] Thibaud Hottelier, Ras Bodik, and Kimiko Ryokai. 2014. Programming by manipulation for layout. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. ACM, Honolulu Hawaii USA, 231–241. `https://doi.org/10.1145/2642918.2647378`

[85] Eric Huang and Richard E. Korf. 2013. Optimal rectangle packing: an absolute placement approach. *J. Artif. Int. Res.* 46, 1 (Jan. 2013), 47–87.

[86] Hua Huang, Lei Zhang, and Hong-Chao Zhang. 2011. Arcimboldo-like collage using internet images. In *Proceedings of the 2011 SIGGRAPH Asia Conference (SA '11)*. Association for Computing Machinery, New York, NY, USA, 1–8. `https://doi.org/10.1145/2024156.2024189`

[87] Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008. Knitting a 3D Model. *Computer Graphics Forum* 27, 7 (Oct. 2008), 1737–1743. `https://doi.org/10.1111/j.1467-8659.2008.01318.x`

[88] Yuki Igarashi and Jun Mitani. 2015. Patchy: an interactive patchwork design system. In *ACM SIGGRAPH 2015 Posters (SIGGRAPH '15)*. Association for Computing Machinery, New York, NY, USA, 1. `https://doi.org/10.1145/2787626.2792601`

[89] Manuel Iori, Vinícius L. de Lima, Silvano Martello, Flávio K. Miyazawa, and Michele Monaci. 2021. Exact solution techniques for two-dimensional cutting and packing. *European Journal of Operational Research* 289, 2 (March 2021), 399–415. `https://doi.org/10.1016/j.ejor.2020.06.050`

[90] Alec Jacobson, Ilya Baran, Jovan Popovic, and Olga Sorkine. 2011. Bounded biharmonic weights for real-time deformation. *ACM Trans. Graph.* 30, 4 (2011), 78.

[91] Alec Jacobson, Daniele Panozzo, et al. 2019. libigl: A simple C++ geometry processing library. https://libigl.github.io/.

[92] Pranjali Jain, Jonathan Balkind, and Timothy Sherwood. 2024. Toxicity versus Carbon Footprint of Computer Systems. In *Proceedings of the Workshop on Hot Topics in Ethical Computer Systems at ASPLOS*. Workshop Publication, San Diego, CA, 1–3.

[93] Pranjali Jain, Claire Pemberton, Ivan Hernandez, Mariana Rosillo, Samantha West, Jonathan Balkind, and Timothy Sherwood. 2024. Electronic Waste Footprint of Computer Systems. In *Proceedings of the Young Architect Workshop at ASPLOS*. Workshop Publication, San Diego, CA, 1–3.

[94] Rijul Jain, Wode Ni, and Joshua Sunshine. 2023. Generating Domain-Specific Programs for Diagram Authoring with Large Language Models. In *Companion Proceedings of the 2023 ACM SIGPLAN International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (SPLASH 2023)*. Association for Computing Machinery, New York, NY, USA, 70–71. `https://doi.org/10.1145/3618305.3623612`

[95] Susmit Jha, Sumit Gulwani, Sanjit A. Seshia, and Ashish Tiwari. 2010. Oracle-guided component-based program synthesis. In *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - Volume 1*. ACM, Cape Town South Africa, 215–224. https://doi.org/10.1145/1806799.1806833

[96] Benjamin Jones, Dalton Hildreth, Duowen Chen, Ilya Baran, Vladimir G. Kim, and Adriana Schulz. 2021. AutoMate: a dataset and learning approach for automatic mating of CAD assemblies. *ACM Transactions on Graphics* 40, 6 (Dec. 2021), 1–18. https://doi.org/10.1145/3478513.3480562

[97] Benjamin Jones, Yuxuan Mei, Haisen Zhao, Taylor Gotfrid, Jennifer Mankoff, and Adriana Schulz. 2021. Computational Design of Knit Templates. *ACM Transactions on Graphics* 41, 2 (Dec. 2021), 16:1–16:16. https://doi.org/10.1145/3488006

[98] R. Kenny Jones, Theresa Barton, Xianghao Xu, Kai Wang, Ellen Jiang, Paul Guerrero, Niloy J. Mitra, and Daniel Ritchie. 2020. ShapeAssembly: learning to generate programs for 3D shape structure synthesis. *ACM Transactions on Graphics* 39, 6 (Dec. 2020), 1–20. https://doi.org/10.1145/3414685.3417812

[99] Jukka Jylänki. 2010. A thousand ways to pack the bin-a practical approach to two-dimensional rectangle bin packing. http://clb.demon.fi/files/RectangleBinPack.pdf

[100] H. Kaebemick, M. Sun, and S. Kara. 2003. Simplified Lifecycle Assessment for the Early Design Stages of Industrial Products. *CIRP Annals* 52, 1 (Jan. 2003), 25–28. https://doi.org/10.1016/S0007-8506(07)60522-8

[101] Karo. 2019. 25 IKEA Hacks to Keep Things Organized. https://craftsyhacks.com/ikea-organizing/

[102] Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. 2019. Knitting Skeletons: A Computer-Aided Design Tool for Shaping and Patterning of Knitted Garments. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology - UIST '19*. ACM Press, New Orleans, LA, USA, 53–65. https://doi.org/10.1145/3332165.3347879

[103] Maeen Md. Khairul Akter, Upama Nasrin Haq, Md. Mazedul Islam, and Mohammad Abbas Uddin. 2022. Textile-apparel manufacturing and material waste management in the circular economy: A conceptual model to achieve sustainable development goal (SDG) 12 for Bangladesh. *Cleaner Environmental Systems* 4 (March 2022), 100070. https://doi.org/10.1016/j.cesys.2022.100070

[104] Sunyoung Kim and Eric Paulos. 2011. Practices in the creative reuse of e-waste. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems (CHI '11)*. Association for Com-

puting Machinery, New York, NY, USA, 2395–2404. https://doi.org/10.1145/1978942.1979292

[105] Donna Kooler. 2012. *Donna Kooler's encyclopedia of knitting* (updated and rev ed.). Leisure Arts, Little Rock, Ark. OCLC: ocn767630287.

[106] Yuki Koyama, Shinjiro Sueda, Emma Steinhardt, Takeo Igarashi, Ariel Shamir, and Wojciech Matusik. 2015. AutoConnect: computational design of 3D-printable connectors. *ACM Transactions on Graphics* 34, 6 (Nov. 2015), 1–11. https://doi.org/10.1145/2816795.2818060

[107] Nikolas Lamb, Sean Banerjee, and Natasha Kholgade Banerjee. 2019. Automated Reconstruction of Smoothly Joining 3D Printed Restorations to Fix Broken Objects. In *Proceedings of the ACM Symposium on Computational Fabrication (SCF '19)*. ACM, New York, NY, USA, 3:1–3:12. https://doi.org/10.1145/3328939.3329005 event-place: Pittsburgh, Pennsylvania.

[108] Mackenzie Leake, Gilbert Bernstein, and Maneesh Agrawala. 2022. Sketch-Based Design of Foundation Paper Pieceable Quilts. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology (UIST '22)*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3526113.3545643

[109] Mackenzie Leake, Gilbert Bernstein, Abe Davis, and Maneesh Agrawala. 2021. A mathematical foundation for foundation paper pieceable quilts. *ACM Transactions on Graphics* 40, 4 (July 2021), 65:1–65:14. https://doi.org/10.1145/3450626.3459853

[110] Mackenzie Leake and Ross Daly. 2024. ScrapMap: Interactive Color Layout for Scrap Quilting. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3654777.3676404

[111] Mackenzie Leake, Frances Lai, Tovi Grossman, Daniel Wigdor, and Ben Lafreniere. 2021. PatchProv: Supporting Improvisational Design Practices for Modern Quilting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3411764.3445601

[112] Catherine Legrand. 2023. *Patchwork: A World Tour*. Thames & Hudson, United States.

[113] Nita Leland. 2011. *New Creative Collage Techniques: How to Make Original Art Using Paper, Color and Texture*. Penguin, United States. Google-Books-ID: hBhjDwAAQBAJ.

[114] Jiahao Li, Meilin Cui, Jeeeun Kim, and Xiang 'Anthony' Chen. 2020. Romeo: A Design Tool for Embedding Transformable Parts in 3D Models to Robotically Augment Default Functionalities. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*. ACM, Virtual Event USA, 897–911. https://doi.org/10.1145/3379337.3415826

[115] Jiahao Li, Jeeeun Kim, and Xiang 'Anthony' Chen. 2019. Robiot: A Design Tool for Actuating Everyday Objects with Automatically Generated 3D Printable Mechanisms. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 673–685. https://doi.org/10.1145/3332165.3347894

[116] Jiahao Li, Alexis Samoylov, Jeeeun Kim, and Xiang 'Anthony' Chen. 2022. Roman: Making Everyday Objects Robotically Manipulable with 3D-Printable Add-on Mechanisms. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 272, 17 pages. https://doi.org/10.1145/3491102.3501818

[117] Yifei Li, David E. Breen, James McCann, and Jessica Hodgins. 2019. Algorithmic Quilting Pattern Generation for Pieced Quilts. In *Proceedings of the 45th Graphics Interface Conference on Proceedings of Graphics Interface 2019 (GI'19)*. Canadian Human-Computer Communications Society, Waterloo, CAN, 1–9. https://doi.org/10.20380/GI2019.13

[118] Bruce Justin Lindbloom. 2017. Delta E (CIE 1994). http://www.brucelindbloom.com/index.html?Eqn_DeltaE_CIE94.html

[119] Chenxi Liu, Jessica Hodgins, and James McCann. 2017. Whole-cloth quilting patterns from photographs. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering (NPAR '17)*. Association for Computing Machinery, New York, NY, USA, 1–8. https://doi.org/10.1145/3092919.3092925

[120] Jingping Liu, Cheng Yang, Haoyi Wu, Ziyin Lin, Zhexu Zhang, Ronghe Wang, Baohua Li, Feiyu Kang, Lei Shi, and Ching Ping Wong. 2014. Future paper based printed circuit boards for green electronics: fabrication and life cycle assessment. *Energy & Environmental Science* 7, 11 (Oct. 2014), 3674–3682. https://doi.org/10.1039/C4EE01995D Publisher: The Royal Society of Chemistry.

[121] Jasmine Lu, Sai Rishitha Boddu, and Pedro Lopes. 2025. ProtoPCB: Reclaiming Printed Circuit Board E-waste as Prototyping Material. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA. https://doi.org/10.1145/3706598.3714095

[122] Jasmine Lu, Beza Desta, K. D. Wu, Romain Nith, Joyce E Passananti, and Pedro Lopes. 2023. ecoEDA: Recycling E-waste During Electronics Design. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3586183.3606745

[123] Jasmine Lu and Pedro Lopes. 2024. Unmaking Electronic Waste. *ACM Trans. Comput.-Hum. Interact.* 31, 6 (Dec. 2024), 77:1–77:30. https://doi.org/10.1145/3674505

[124] Makersite. 2024. Makersite. https://makersite.io/

[125] Ilan Mandel and Wendy Ju. 2023. Recapturing Product as Material Supply: Hoverboards as Garbatrage. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference (DIS '23)*. Association for Computing Machinery, New York, NY, USA, 564–579. https://doi.org/10.1145/3563657.3596128

[126] Ilan Mandel and Wendy Ju. 2024. Designing with What Remains. In *Proceedings of the 2024 ACM Designing Interactive Systems Conference (DIS '24)*. Association for Computing Machinery, New York, NY, USA, 3002–3015. https://doi.org/10.1145/3643834.3661628

[127] James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jen Mankoff, and Jessica Hodgins. 2016. A compiler for 3D machine knitting. *ACM Transactions on Graphics* 35, 4 (July 2016), 1–11. https://doi.org/10.1145/2897824.2925940

[128] Yuxuan Mei, Benjamin Jones, Dan Cascaval, Jennifer Mankoff, Etienne Vouga, and Adriana Schulz. 2024. FabHacks: Transform Everyday Objects into Home Hacks Leveraging a Solver-aided DSL. In *Proceedings of the 9th ACM Symposium on Computational Fabrication (SCF '24)*. Association for Computing Machinery, New York, NY, USA, 1–16. https://doi.org/10.1145/3639473.3665788

[129] Yuxuan Mei, Lirong Que, Mackenzie Leake, and Adriana Schulz. 2025. ScrapPack: Material-Aware Patchwork Design using Scrap Fabrics. Under Submission. Manuscript available upon request..

[130] Musashi Shinjo, Maria Larsson, and Hironori Yoshida. 2024. A Design and Fabrication Workflow for Upcycling Leftover Fabrics as Mosaic Art. https://computationalcreativity.net/iccc24/papers/ICCC24_paper_153.pdf

[131] Mxbonn. 2025. Mxbonn/strip-packing: Algorithm for the strip packing problem with guillotine cuts constraint. https://github.com/Mxbonn/strip-packing

[132] Tomás Méndez Echenagucia, Alfonso Capozzoli, Ylenia Cascone, and Mario Sassone. 2015. The early design stage of a building envelope: Multi-objective search through heating, cooling and lighting energy performance analysis. *Applied Energy* 154 (Sept. 2015), 577–591. https://doi.org/10.1016/j.apenergy.2015.04.090

[133] Tomás Méndez Echenagucia, Teresa Moroseos, and Christopher Meek. 2023. On the tradeoffs between embodied and operational carbon in building envelope design: The impact of local climates and energy grids. *Energy and Buildings* 278 (Jan. 2023), 112589. https://doi.org/10.1016/j.enbuild.2022.112589

[134] Samuel Naffziger, Noah Beck, Thomas Burd, Kevin Lepak, Gabriel H. Loh, Mahesh Subramony, and Sean White. 2021. Pioneering chiplet technology and design for the AMD EPYC™ and Ryzen™

processor families. In *Proceedings of the 48th Annual International Symposium on Computer Architecture (ISCA '21)*. IEEE Press, Virtual Event, Spain, 57–70. https://doi.org/10.1109/ISCA52012.2021.00014

[135] Chandrakana Nandi, James R Wilcox, Pavel Panchekha, Taylor Blau, Dan Grossman, and Zachary Tatlock. 2018. Functional programming for compiling and decompiling computer-aided design. *Proceedings of the ACM on Programming Languages* 2, ICFP (2018), 1–31.

[136] Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James Mccann. 2018. Automatic Machine Knitting of 3D Meshes. *ACM Transactions on Graphics* 37, 3 (Aug. 2018), 1–15. https://doi.org/10.1145/3186265

[137] Vidya Narayanan, Kui Wu, Cem Yuksel, and James McCann. 2019. Visual knitting machine programming. *ACM Transactions on Graphics* 38, 4 (July 2019), 1–13. https://doi.org/10.1145/3306346.3322995

[138] N. F. Nissen, H. Griese, A. Middendorf, J. Müller, H. Pötter, and H. Reichl. 1997. Comparison of simplified environmental assessments versus full life cycle assessment (LCA) for the electronics designer. In *Life Cycle Networks: Proceedings of the 4th CIRP International Seminar on Life Cycle Engineering 26–27 June 1997, Berlin, Germany*, F.-L. Krause and G. Seliger (Eds.). Springer US, Boston, MA, 301–312. https://doi.org/10.1007/978-1-4615-6381-5_25

[139] Jorge Nocedal and Stephen J. Wright. 2006. *Numerical Optimization* (2e ed.). Springer, New York, NY, USA.

[140] Olimex. 2020. Olimex DIY-Laptop: TERES-PCB1-A64-MAIN PCB Rev.C. https://github.com/OLIMEX/DIY-LAPTOP/tree/master

[141] Elsa Olivetti and Randolph Kirchain. 2012. A Product Attribute to Impact Algorithm to Streamline IT Carbon Footprinting. In *Design for Innovative Value Towards a Sustainable Society*, Mitsutaka Matsumoto, Yasushi Umeda, Keijiro Masui, and Shinichi Fukushige (Eds.). Springer Netherlands, Dordrecht, 747–749. https://doi.org/10.1007/978-94-007-3010-6_151

[142] Onshape. 2023. Onshape | Product Development Platform. https://www.onshape.com/en/

[143] OpenAI. 2024. ChatGPT. https://chat.openai.com

[144] Bridget O'Flaherty. 2023. Sustainable Quilting – Part Three. https://canadianquilter.com/sustainable-quilting-part-three/

[145] Daniele Panozzo, Enrico Puppo, Marco Tarini, and Olga Sorkine-Hornung. 2014. Frame fields: anisotropic and non-orthogonal cross fields. *ACM Trans. Graph.* 33, 4, Article 134 (July 2014), 11 pages. https://doi.org/10.1145/2601097.2601179

[146] Victor Papanek. 2019. *Design for the Real World* (3 ed.). Thames & Hudson, London.

[147] Pratyush Patel, Theo Gregersen, and Thomas Anderson. 2023. An Agile Pathway Towards Carbon-aware Clouds. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems* (Boston, MA, USA) *(HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, Article 10, 8 pages. https://doi.org/10.1145/3604930.3605722

[148] Phitchaya Mangpo Phothilimthana, Archibald Samuel Elliott, An Wang, Abhinav Jangda, Bastian Hagedorn, Henrik Barthels, Samuel J. Kaufman, Vinod Grover, Emina Torlak, and Rastislav Bodik. 2019. Swizzle Inventor: Data Movement Synthesis for GPU Kernels. In *Proceedings of the Twenty-Fourth International Conference on Architectural Support for Programming Languages and Operating Systems*. ACM, Providence RI USA, 65–78. https://doi.org/10.1145/3297858.3304059

[149] Christina Piereder, Günter Fleck, Verena Geist, Michael Moser, and Josef Pichler. 2024. Using AI-Based Code Completion for Domain-Specific Languages. In *Product-Focused Software Process Improvement (Lecture Notes in Computer Science)*, Regine Kadgien, Andreas Jedlitschka, Andrea Janes, Valentina Lenarduzzi, and Xiaozhou Li (Eds.). Springer Nature Switzerland, Cham, 227–242. https://doi.org/10.1007/978-3-031-49266-2_16

[150] Enrico Pietrobuoni. 2015. *Two-Dimensional Bin Packing Problem with Guillotine Restrictions*. Ph. D. Dissertation. University of Bologna. https://api.semanticscholar.org/CorpusID:46201745

[151] Lauren Piro. 2015. 8 Clutter Problems Solved by Shower Rings. https://www.goodhousekeeping.com/home/decorating-ideas/shower-curtain-rings-organizing

[152] M. Popescu, M. Rippmann, T. Van Mele, and P. Block. 2017. Automated generation of knit patterns for non-developable surfaces. In *Humanizing Digital Reality - Proceedings of the Design Modelling Symposium 2017*, K. De Rycke et al. (Ed.). Springer, Paris, 271–284. https://doi.org/10.1007/978-981-10-6611-5_24

[153] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 2016. RetroFab: A Design Tool for Retrofitting Physical Interfaces using Actuators, Sensors and 3D Printing. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*. ACM, San Jose California USA, 409–419. https://doi.org/10.1145/2858036.2858485

[154] Ravelry. 2019. Ravelry. https://ravelry.com/

[155] Nicolas Ray, Bruno Vallet, Wan Chiu Li, and Bruno Lévy. 2008. N-symmetry direction field design. *ACM Transactions on Graphics* 27, 2 (April 2008), 1–13. https://doi.org/10.1145/1356682.1356683

[156] Bernhard Reinert, Tobias Ritschel, and Hans-Peter Seidel. 2013. Interactive by-example design of artistic packing layouts. *ACM Trans. Graph.* 32, 6 (Nov. 2013), 218:1–218:7. https://doi.org/10.1145/2508363.2508409

[157] T. I. Rissanen. 2013. *Zero-waste fashion design : a study at the intersection of cloth, fashion design and pattern cutting*. Thesis. University of Technology Sydney. https://opus.lib.uts.edu.au/handle/10453/23384 Accepted: 2013-06-28T05:40:14Z.

[158] Carsten Rother, Lucas Bordeaux, Youssef Hamadi, and Andrew Blake. 2006. AutoCollage. *ACM Trans. Graph.* 25, 3 (July 2006), 847–852. https://doi.org/10.1145/1141911.1141965

[159] Ryan Schmidt and Nobuyuki Umetani. 2014. Branching support structures for 3D printing. In *ACM SIGGRAPH 2014 Studio (SIGGRAPH '14)*. Association for Computing Machinery, New York, NY, USA, 1. https://doi.org/10.1145/2619195.2656293

[160] Tyler Schroeder. 2024. tyschroed/guillotine-packer. https://github.com/tyschroed/guillotine-packer original-date: 2020-01-26T05:58:32Z.

[161] Adriana Schulz, Ariel Shamir, Ilya Baran, David I. W. Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. 2017. Retrieval on Parametric Shape Collections. *ACM Trans. Graph.* 36, 1, Article 11 (Jan. 2017), 14 pages. https://doi.org/10.1145/2983618

[162] Adriana Schulz, Ariel Shamir, David IW Levin, Pitchaya Sitthi-Amorn, and Wojciech Matusik. 2014. Design and fabrication by example. *ACM Transactions on Graphics (TOG)* 33, 4 (2014), 1–11.

[163] Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. 2017. Interactive design space exploration and optimization for CAD models. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–14.

[164] secnot. 2025. secnot/rectpack. https://github.com/secnot/rectpack original-date: 2016-12-09T08:17:38Z.

[165] Nicholas Sharp et al. 2019. Polyscope. www.polyscope.run.

[166] Elena Shcherban. 2023. Fairphone 4 New Life Edition - refurbished smartphone with 5-year warranty and updates until 2028. https://gagadget.com/en/announce/356350-fairphone-4-new-life-edition-refurbished-smartphone-with-5-year-warranty-and-updates-until-2028/

[167] SHIMA SEIKI. 2019. SDS-ONE APEX series. https://www.shimaseiki.com/product/design/

[168] Kamyar Shirvanimoghaddam, Bahareh Motamed, Seeram Ramakrishna, and Minoo Naebe. 2020. Death by waste: Fashion and textile circular economy case. *Science of The Total Environment* 718 (May 2020), 137317. https://doi.org/10.1016/j.scitotenv.2020.137317

[169] Maria Shugrina, Ariel Shamir, and Wojciech Matusik. 2015. Fab forms: Customizable objects for fabrication with validity and geometry caching. *ACM Transactions on Graphics (TOG)* 34, 4 (2015), 1–12.

[170] Marta Skreta, Naruki Yoshikawa, Sebastian Arellano-Rubach, Zhi Ji, Lasse Bjørn Kristensen, Kourosh Darvish, Alán Aspuru-Guzik, Florian Shkurti, and Animesh Garg. 2023. Errors are Useful Prompts: Instruction Guided Task Programming with Verifier-Assisted Iterative Prompting. https://doi.org/10.48550/arXiv.2303.14100 arXiv:2303.14100 [cs].

[171] Sluicebox. 2024. Sluicebox. https://www.sluicebox.ai/

[172] SOLIDWORKS. 2023. 3D CAD Design Software | SOLIDWORKS. https://www.solidworks.com/

[173] Katherine W. Song, Samar Sabie, Steven J. Jackson, Kristina Lindström, Eric Paulos, Åsa Ståhl, and Ron Wakkary. 2025. Unmaking and HCI: Techniques, Technologies, Materials, and Philosophies beyond Making. *ACM Trans. Comput.-Hum. Interact.* 31, 6 (Jan. 2025), 76:1–76:6. https://doi.org/10.1145/3689047

[174] David J. Spencer. 2001. *Knitting technology : a comprehensive handbook and practical guide*. Technomic, Lancaster, Pennsylvania. 386 pages.

[175] Sphera. 2024. Life Cycle Assessment Software and Data | Sphera (GaBi). https://sphera.com/solutions/product-stewardship/life-cycle-assessment-software-and-data/

[176] Jenny Stanley. 2021. 33 Brilliant Home Hacks Using Our 3 Favorite Items. https://www.familyhandyman.com/list/20-home-hacks-hangers-rubber-bands-and-cardboard-tubes/

[177] Francesca Stazi, Alessio Mastrucci, and Placido Munafò. 2012. Life cycle assessment approach for the optimization of sustainable building envelopes: An application on solar wall systems. *Building and Environment* 58 (Dec. 2012), 278–288. https://doi.org/10.1016/j.buildenv.2012.08.003

[178] Martha Stewart. 2024. How to Upcycle Every Type of Fabric Scrap. https://www.marthastewart.com/8263546/how-upcycle-every-fabric-scrap-type Section: Martha Stewart.

[179] STOLL. 2019. M1PLUS®. https://www.stoll.com/en/software/m1plusr/

[180] Sascia Strohhammer. 2021. *Sustainable Knitting for Beginners and Beyond: 20 Patterns for Environmentally Friendly Knits*. David and Charles, England.

[181] Karen Sullivan and Jim Heumann. 2019. Karen and Jim's Excellent Adventure: Fiddly Bits: Making life on a small boat safer and more comfortable. http://karenandjimsexcellentadventure.blogspot.com/p/fiddly-bits.html

[182] Timna Tarr. 2022. Stitched Photo Mosaic Quilting. https://www.connectingthreads.com/stitched-photo-mosaic-quilting/p/47031

[183] Alexander Teibrich, Stefanie Mueller, FranÃ§ois GuimbretiÃšre, Robert Kovacs, Stefan Neubert, and Patrick Baudisch. 2015. Patching Physical Objects. In *Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology (UIST '15)*. ACM, New York, NY, USA, 83–91. https://doi.org/10.1145/2807442.2807467 event-place: Charlotte, NC, USA.

[184] Emina Torlak and Rastislav Bodik. 2013. Growing Solver-Aided Languages with Rosette. In *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software* (Indianapolis, Indiana, USA) *(Onward! 2013)*. Association for Computing Machinery, New York, NY, USA, 135–152. https://doi.org/10.1145/2509578.2509586

[185] Climate Action Tracker. 2024. Warming Projections Global Update. https://climateactiontracker.org/documents/1277/CAT_2024-11-14_GlobalUpdate_COP29.pdf

[186] Climate Action Tracker. 2025. RELEASE: The climate is warming and sea levels rising way faster than governments are acting. https://climateactiontracker.org/press/release-the-climate-is-warming-and-sea-levels-rising-way-faster-than-governments-are-acting/

[187] Jenny Underwood. 2009. *The design of 3D shape knitted preforms*. Ph. D. Dissertation. RMIT University.

[188] UNFCCC. 2015. The Paris Agreement | UNFCCC. https://unfccc.int/process-and-meetings/the-paris-agreement

[189] Vera Vandenbosch. 2015. *Scraps: stylish stash fabric crafts to stitch*. The Taunton Press, Newtown, CT. OCLC: 881041289.

[190] J. Vanek, J. a. G. Galicia, and B. Benes. 2014. Clever Support: Efficient Support Structure Generation for Digital Fabrication. *Computer Graphics Forum* 33, 5 (2014), 117–125. https://doi.org/10.1111/cgf.12437 _eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.12437.

[191] Eldy S. Lazaro Vasquez and Katia Vega. 2019. Myco-accessories: sustainable wearables with biodegradable materials. In *Proceedings of the 2019 ACM International Symposium on Wearable Computers (ISWC '19)*. Association for Computing Machinery, New York, NY, USA, 306–311. https://doi.org/10.1145/3341163.3346938

[192] Amir Vaxman et al. 2020. Directional: A library for Directional Field Synthesis, Design, and Processing. https://doi.org/10.5281/zenodo.3338174

[193] Amir Vaxman, Marcel Campen, Olga Diamanti, Daniele Panozzo, David Bommes, Klaus Hildebrandt, and Mirela Ben-Chen. 2016. Directional Field Synthesis, Design, and Processing. *Computer Graphics Forum* 35, 2 (May 2016), 545–572. https://doi.org/10.1111/cgf.12864

[194] VentureWell. 2023. Measuring Sustainability. https://venturewell.org/tools_for_design/measuring-sustainability/life-cycle-assessment-content/

[195] Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. 2020. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods* 17 (2020), 261–272. https://doi.org/10.1038/s41592-019-0686-2

[196] Domokos Váncsa. 2025. "Fabric" Category - Seamless Textures. https://www.tilingtextures.com/textures/category/materials/fabric/

[197] Ludwig Wilhelm Wall, Alec Jacobson, Daniel Vogel, and Oliver Schneider. 2021. Scrappy: Using Scrap Material as Infill to Make Fabrication More Sustainable. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems (CHI '21)*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3411764.3445187

[198] Jaylen Wang, Udit Gupta, and Akshitha Sriraman. 2023. Peeling Back the Carbon Curtain: Carbon Optimization Challenges in Cloud Computing. In *Proceedings of the 2nd Workshop on Sustainable Computer Systems (HotCarbon '23)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3604930.3605718

[199] Jingdong Wang, Long Quan, Jian Sun, Xiaoou Tang, and Heung-Yeung Shum. 2006. Picture Collage. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, Vol. 1. IEEE, New York, NY, USA, 347–354. https://doi.org/10.1109/CVPR.2006.224 ISSN: 1063-6919.

[200] Radha Weaver. 2025. Thrift Your Fabric. https://www.sewingthroughfog.com/thriftyourfabric

[201] Gregor Wernet, Christian Bauer, Bernhard Steubing, Jürgen Reinhard, Emilia Moreno-Ruiz, and Bo Weidema. 2016. The ecoinvent database version 3 (part I): overview and methodology. *The International Journal of Life Cycle Assessment* 21, 9 (Sept. 2016), 1218–1230. https://doi.org/10.1007/s11367-016-1087-8

[202] Chenming Wu, Haisen Zhao, Chandrakana Nandi, Jeffrey I. Lipton, Zachary Tatlock, and Adriana Schulz. 2019. Carpentry compiler. *ACM Trans. Graph.* 38, 6 (Nov. 2019), 195:1–195:14. https://doi.org/10.1145/3355089.3356518

[203] Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. 2018. Stitch meshing. *ACM Transactions on Graphics* 37, 4 (July 2018), 1–14. https://doi.org/10.1145/3197517.3201360

[204] Kui Wu, Hannah Swan, and Cem Yuksel. 2019. Knittable Stitch Meshes. *ACM Transactions on Graphics* 38, 1 (Jan. 2019), 1–13. https://doi.org/10.1145/3292481

[205] Shanel Wu and Laura Devendorf. 2020. Unfabricate: Designing Smart Textiles for Disassembly. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3313831.3376227

[206] Yang Xu, Ziqi Wang, Siyu Gong, and Yong Chen. 2021. Reusable support for additive manufacturing. *Additive Manufacturing* 39 (March 2021), 101840. https://doi.org/10.1016/j.addma.2021.101840

[207] Zeyu Yan, Tingyu Cheng, Jasmine Lu, Pedro Lopes, and Huaishu Peng. 2023. Future Paradigms for Sustainable Making. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23 Adjunct)*. Association for Computing Machinery, New York, NY, USA, 1–3. https://doi.org/10.1145/3586182.3617433

[208] Zeyu Yan, Mrunal Sanjay Dhaygude, and Huaishu Peng. 2025. Make Making Sustainable: Exploring Sustainability Practices, Challenges, and Opportunities in Making Activities. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3706598.3713665

[209] Zeyu Yan, Jiasheng Li, Zining Zhang, and Huaishu Peng. 2024. SolderlessPCB: Reusing Electronic Components in PCB Prototyping through Detachable 3D Printed Housings. In *Proceedings of the CHI Conference on Human Factors in Computing Systems* (Honolulu, HI, USA) *(CHI '24)*. Association for Computing Machinery, New York, NY, USA, Article 345, 17 pages. https://doi.org/10.1145/3613904.3642765

[210] Zeyu Yan, Advait Vartak, Jiasheng Li, Zining Zhang, and Huaishu Peng. 2025. PCB Renewal: Iterative Reuse of PCB Substrates for Sustainable Electronic Making. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, 1–18. https://doi.org/10.1145/3706598.3714276

[211] Evan Yares. 2013. The failed promise of parametric CAD part 1: From the beginning. https://www.3dcadworld.com/the-failed-promise-of-parametric-cad/. (Accessed on 09/06/2019).

[212] Katherine Ye, Wode Ni, Max Krieger, Dor Ma'ayan, Jenna Wise, Jonathan Aldrich, Joshua Sunshine, and Keenan Crane. 2020. Penrose: from mathematical notation to beautiful diagrams. *ACM Transactions on Graphics* 39, 4 (Aug. 2020), 144:144:1–144:144:16. https://doi.org/10.1145/3386569.3392375

[213] Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2012. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Transactions on Graphics* 31, 4 (July 2012), 1–12. https://doi.org/10.1145/2185520.2185533

[214] Cheng Zhang, Yu Zheng, Junfeng Jing, Yun Liu, and Haihong Huang. 2022. A comparative LCA study on aluminum electrolytic capacitors: From liquid-state electrolyte, solid-state polymer to their hybrid. *Journal of Cleaner Production* 375 (Nov. 2022), 134044. https://doi.org/10.1016/j.jclepro.2022.134044

[215] Defu Zhang, Leyuan Shi, Stephen C. H. Leung, and Tao Wu. 2016. A priority heuristic for the guillotine rectangular packing problem. *Inform. Process. Lett.* 116, 1 (Jan. 2016), 15–21. https://doi.org/10.1016/j.ipl.2015.08.008

[216] Ruowang Zhang, Stefanie Mueller, Gilbert Louis Bernstein, Adriana Schulz, and Mackenzie Leake. 2024. WasteBanned: Supporting Zero Waste Fashion Design Through Linked Edits. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3654777.3676395

[217] Zhihan Zhang, Agni K. Biswal, Ankush Nandi, Kali Frost, Jake A. Smith, Bichlien H. Nguyen, Shwetak Patel, Aniruddh Vashisth, and Vikram Iyer. 2024. Recyclable vitrimer-based printed circuit boards for sustainable electronics. *Nature Sustainability* 7, 5 (May 2024), 616–627. https://doi.org/10.1038/s41893-024-01333-7 Publisher: Nature Publishing Group.

[218] Zhihan Zhang, Felix Hähnlein, Yuxuan Mei, Zachary Englhardt, Shwetak Patel, Adriana Schulz, and Vikram Iyer. 2024. DeltaLCA: Comparative Life-Cycle Assessment for Electronics Design. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 8, 1 (March 2024), 29:1–29:29. https://doi.org/10.1145/3643561

[219] Allan Zhao, Jie Xu, Mina Konaković-Luković, Josephine Hughes, Andrew Spielberg, Daniela Rus, and Wojciech Matusik. 2020. RoboGrammar: graph grammar for terrain-optimized robot design. *ACM Transactions on Graphics* 39, 6 (Dec. 2020), 1–16. `https://doi.org/10.1145/3414685.3417831`

[220] Haisen Zhao, Max Willsey, Amy Zhu, Chandrakana Nandi, Zachary Tatlock, Justin Solomon, and Adriana Schulz. 2022. Co-Optimization of Design and Fabrication Plans for Carpentry. *ACM Transactions on Graphics* 41, 3 (March 2022), 32:1–32:13. `https://doi.org/10.1145/3508499`

[221] Amy Zhu, Yuxuan Mei, Benjamin Jones, Zachary Tatlock, and Adriana Schulz. 2024. Computational Illusion Knitting. *ACM Trans. Graph.* 43, 4 (July 2024), 152:1–152:13. `https://doi.org/10.1145/3658231`

Chapter A

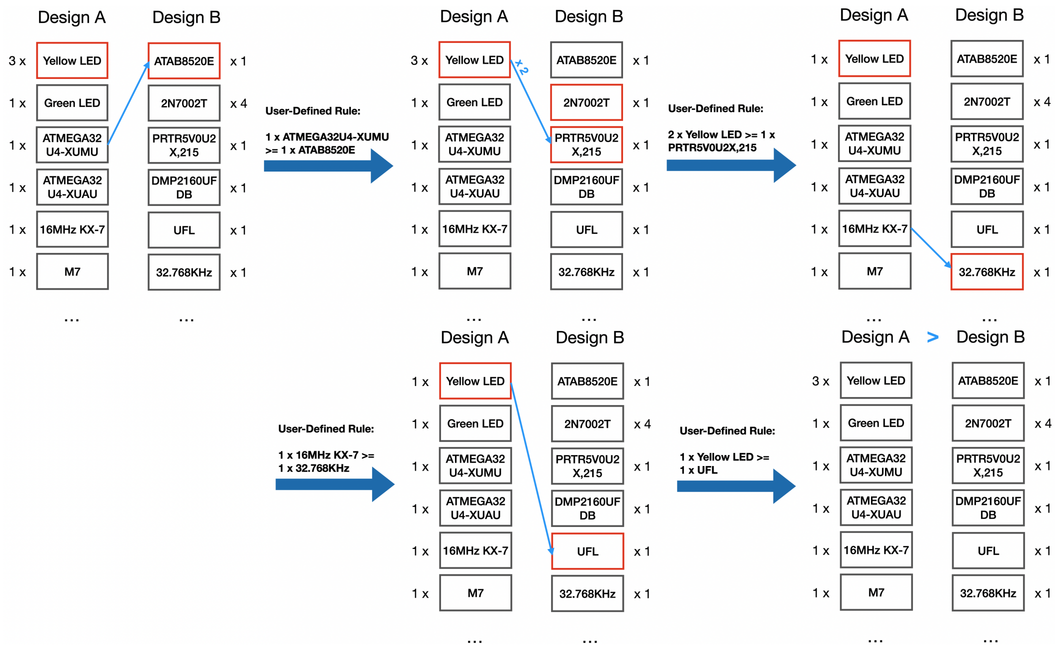
Appendix

A.1 DeltaLCA Case Study Summary

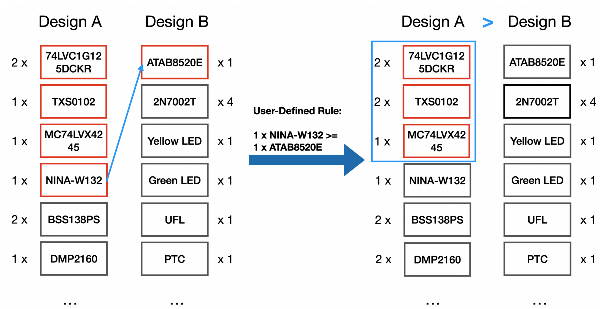
We summarized the basic attributes of the three Arduino dev boards and user interaction with our tool for comparing the EIs of these devices in the case study.

	Board	Board Area (mm^2)	#Layers	#Components	#ICs from Automated LCI
(1)	Leonardo R3	4180.6368	2	97	20
(2)	MKR FOX 1200	1696.76	4	92	20
(3)	UNO WiFi	3659.124	2	117	26

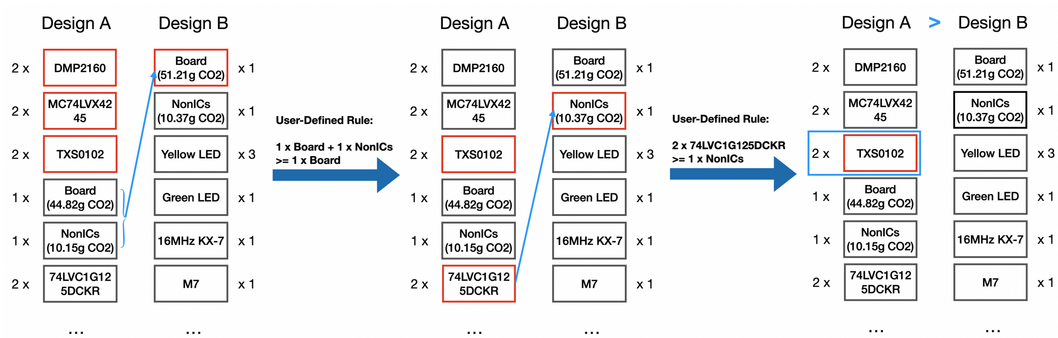
Table A.1: Basic attributes of the three Arduino dev boards.



(a) Arduino Leonardo R3 ≥ Arduino MKR FOX 1200



(b) Arduino UNO WiFi ≥ Arduino Leonardo R3



(c) Arduino UNO WiFi ≥ Arduino MKR FOX 1200

Figure A.1: Summary of the user interactions with the UI for comparing the three boards. In the subfigures, the red boxes represent unmatched components, and the gray boxes represent matched components. The blue arrow indicates a user-defined rule between components in design A and design B. Here the user is proving A > B for all three cases, so they stop when all components in B are matched and zero or more components in A are unmatched (indicated with the blue box in the last step).

A.2 Analysis of the Design Space of Home Hacks

The concept of "home hacking" covers a variety of topics. To better understand this design space, we analyzed a collection of hacks and defined our problem domain, which informed our DSL design.

We first gathered over 400 examples of hacks across 17 sources (including DIY blogs, videos, and individual designs from colleagues). After eliminating hacks that are repeated, or essentially the same but used under different scenarios, we selected 48 distinct hacks for further analysis.

Our analysis started with identifying each hack's functionality. This gives us two main categories: hacks that hold or fix some objects in a specific location and orientation – or *fixtures* (27 out of 48), and hacks which typically make creative reuse of a single item to change the shape or feel (material property) of an existing object to allow for better grasping or easier interaction, e.g., a pool noodle used to organize wires and hide them, cover sharp saw edges, or create padding on furniture corners for a baby-proof environment. In contrast to the latter category, fixture hacks usually involve multiple parts, e.g., several wire baskets chained together with S-hooks to organize items above a kitchen sink. And since their goal is to hold a part at a specific location and orientation relative to its environment, gravity will affect the design's stability. Thus, fixture design can be hard to reason with intuition and is well-suited as a computational design problem. Since deformable objects are difficult to model and simulate efficiently compared to rigid bodies, and it is unclear how end users can accurately specify manual modifications, we further limit the domain to rigid fixtures with only undeformed constituting objects because they are the majority of fixtures and present a well-scoped subset (24 out of 27). We show the complete set of 24 rigid (or can be seen as rigid) fixture hacks in Figure A.2.

We then analyzed how the individual objects, which we call "parts", were connected in the subset of rigid undeformed fixtures. Although many different parts are involved in the hack examples, connections typically form between common types of *connector primitives*. These connector primitives connect in ways that are independent of the objects that they are part of. From our analysis, we extract the following categories of connector primitives (Figure 4.16): *rod*, *hook*, *ring*, *tube*, *clip*, *edge*, *surface*, and *hemisphere*.

Further analyzing the hack designs by looking at the parts and the connecting shapes of the parts, we determined the relationship between these connector primitives on whether they connect and how they align with each other when they connect. We summarized the common design patterns of how objects can be connected in Table A.3. The eight connector primitives and their pairwise interactions become the basis for the design of the DSL and the underlying logic for assembly building in our system (see Section 4.2.4).

A.3 FabHaL Programs for Gallery Examples

A.3.1 Toothbrush Holder

```
ASSEMBLY_toothbrush_holder = Assembly()

surface = Surface({"width": 400, "length": 400})
ENV_start = Environment({"surface": surface})
ENV_end = Environment({"toothbrush": Toothbrush()})
```

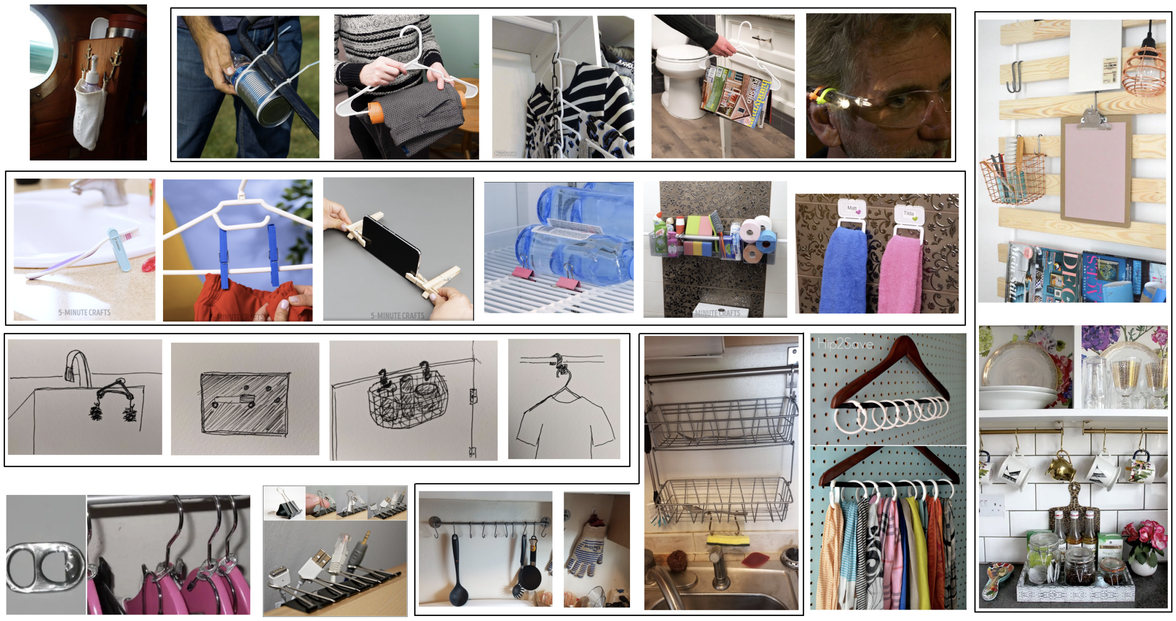


Figure A.2: As ordered in the image, the hacks or hack groups in black boxes are from [181, 176, 40], one of the colleagues, [29, 67], another one of the colleagues, [151, 101]. We provide a short description for each hack and its source in Table A.2.

```
start_frame = Frame()
ASSEMBLY_toothbrush_holder.start_with(ENV_start, start_frame)

PART_clip = PlasticClip()
ASSEMBLY_toothbrush_holder.connect(PART_clip.hemisphere1, ENV_start.surface)
ASSEMBLY_toothbrush_holder.connect(PART_clip.hemisphere2, ENV_start.surface)
ASSEMBLY_toothbrush_holder.connect(ENV_end.rod, PART_clip.clip)
ASSEMBLY_toothbrush_holder.connect(ENV_end.hemisphere, ENV_start.surface)

end_frame = Frame([-10, -62.5, 50], [-65,0,0])
ASSEMBLY_toothbrush_holder.end_with(ENV_end, end_frame)
```

A.3.2 Charger Holder

```
ASSEMBLY_cable_holder = Assembly()

edge = Edge({"width": 100, "length": 200, "height": 1.5})
ENV_start = Environment({"edge": edge})
ENV_end = Environment({"cable": Cable()})

start_frame = Frame([0,0,150],[0,0,0])
ASSEMBLY_cable_holder.start_with(ENV_start.edge, start_frame)

PART_binderclip = BinderClip()
ASSEMBLY_cable_holder.connect(PART_binderclip.clip, ENV_start.edge, is_fixed=True)
ASSEMBLY_cable_holder.connect(ENV_end.rod1, PART_binderclip.ring1)
ASSEMBLY_cable_holder.connect(ENV_end.rod1, PART_binderclip.ring2)

end_frame = Frame([0,57.5,163], [0,0,0])
ASSEMBLY_cable_holder.end_with(ENV_end.rod2, end_frame)
```

Table A.2: Hacks and their sources.

No.	Hack Source	Short Description
1	[181]	soap bottle bag
2	[176]	bottle holder on mower
3		nonslip hanger
4		hangers chained with rings
5		magazine on a hanger
6		glass light holder
7	[40]	toothbrush holder
8		pants hanger with clothes clips
9		phone holder from clothes clips
10		binder clips stoppers in fridge
11		bathroom organizer
12		tissue box towel hanger
13	Colleague A	dish sponge hanger
14		pen over pins on a board
15		shower essentials holder
16		hang clothes parallel to wall
17	[29]	hangers chained with soda can tabs
18	[67]	charger holder
19	Colleague B	kitchen tools rack
20		oven mitten holder
21		wire baskets chained with S-hooks
22	[151]	scarf organizer
23	[101]	bed slat as rack
24		cup hanger

Table A.3: Number of appearances in the 24 rigid undeformed fixture hacks (Figure A.2) of each connection type. * means this connection type didn't appear but we deduced that it is compatible based on similar connection types. # means this connection type didn't appear in rigid fixture hacks but appeared in a non-rigid fixture hack. Light grey means this connection type did not appear or cannot be deduced. We ignore the lower-triangular region (dark grey) as it is redundant with the upper-triangular one.

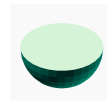
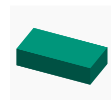
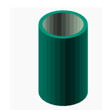
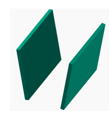
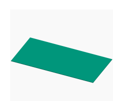
	hook	ring	hemi.	edge	rod	tube	clip	surf.
hook	1	5			12	1		
ring					4	*		
hemi.								2
edge							3	
rod						2	5	
tube						#	*	
clip								
surf.								7

A.3.3 Soap Bottle Holder

```
ASSEMBLY_soapbottle_holder = Assembly()

rod = Rod({"length": 500, "radius": 5})
ENV_start = Environment({"door": rod})
```

Table A.4: We parametrize each primitive with its corresponding shape parameters, and show an example of the primitive.

Primitive	Shape Parameters	Example
hook	arc angle, arc radius, thickness	
ring	arc radius, thickness	
hemisphere	radius	
edge	width, length, height	
rod	radius, length	
tube	inner radius, thickness, length	
clip	width, height, base distance, open gap, thickness	
surface	width, length	

```
ENV_end = Environment({"soapbottle": SoapBottle()})

start_frame = Frame([0,0,500], [90,0,90])
ASSEMBLY_soapbottle_holder.start_with(ENV_start.door, start_frame)

PART_hookeye1 = HookEyeLeftS()
ASSEMBLY_soapbottle_holder.connect(PART_hookeye1.ring, ENV_start.door)
PART_basket = Basket()
ASSEMBLY_soapbottle_holder.connect(PART_basket.rod1, PART_hookeye1.hook)
PART_hookeye2 = HookEyeLeftS()
ASSEMBLY_soapbottle_holder.connect(PART_hookeye2.ring, ENV_start.door, alignment="flip")
ASSEMBLY_soapbottle_holder.connect(PART_hookeye2.hook, PART_basket.rod2)
ASSEMBLY_soapbottle_holder.connect(ENV_end.surface, PART_basket.surface)

end_frame = Frame([0,0,253], [0,0,180])
ASSEMBLY_soapbottle_holder.end_with(ENV_end, end_frame)
```

A.3.4 Mug Hanger

```
ASSEMBLY_mug_hanger = Assembly()

rod = Rod({"length": 500, "radius": 2})
ENV_start = Environment({"rod": rod})
```

```
surface = Surface({"length": 800, "width": 600})
ENV_wall = Environment({"wall": surface})
ENV_end = Environment({"mug": Mug()})

start_frame = Frame([0,0,200], [90,0,90])
ASSEMBLY_mug_hanger.start_with(ENV_start.rod, start_frame)
wall_frame = Frame([0,50,0], [90,0,0])
ASSEMBLY_mug_hanger.start_with(ENV_wall.wall, wall_frame)

PART_doublehook1 = DoubleHook()
PART_doublehook2 = DoubleHook()
PART_doublehook3 = DoubleHook()
ASSEMBLY_mug_hanger.connect(PART_doublehook1.hook2, ENV_start.rod)
ASSEMBLY_mug_hanger.connect(PART_doublehook2.hook2, PART_doublehook1.hook1)
ASSEMBLY_mug_hanger.connect(PART_doublehook3.hook1, PART_doublehook2.hook1)
ASSEMBLY_mug_hanger.connect(ENV_end.hook, PART_doublehook3.hook2)

end_frame = Frame([0,0,50], [-35,0,-90])
ASSEMBLY_mug_hanger.end_with(ENV_end.hook, end_frame)
```

A.3.5 Paper Towel Holder

```
ASSEMBLY_paper_towel_holder = Assembly()

ENV_start = Environment({"env": TowelHangingEnv()})
ENV_end = Environment({"paper_towel_roll": PaperTowelRoll()})

wall_frame = Frame([0,0,300], [0,0,0])
ASSEMBLY_paper_towel_holder.start_with(ENV_start, wall_frame)

PART_hookeye1 = HookEyeLeft()
PART_hookeye2 = HookEyeLeft()
PART_broomrod = BroomRod()
ASSEMBLY_paper_towel_holder.connect(PART_hookeye1.ring, ENV_start.hook1)
ASSEMBLY_paper_towel_holder.connect(PART_hookeye2.ring, ENV_start.hook2)
ASSEMBLY_paper_towel_holder.connect(PART_broomrod.tube, PART_hookeye1.hook, is_fixed=True)
ASSEMBLY_paper_towel_holder.connect(ENV_end.tube, PART_broomrod.tube)
ASSEMBLY_paper_towel_holder.connect(PART_hookeye2.hook, PART_broomrod.tube, is_fixed=True)

end_frame = Frame([53,0,160], [-90,-60,0])
ASSEMBLY_paper_towel_holder.end_with(ENV_end.tube, end_frame)
```

A.3.6 Diaper Caddy

```
ASSEMBLY_diaper_caddy = Assembly()

ENV_start = Environment({"backseat": BackSeats()})
ENV_end = Environment({"diaper_caddy": DiaperCaddy()})

start_frame = Frame([0,0,0], [0,0,0])
ASSEMBLY_diaper_caddy.start_with(ENV_start, start_frame)

PART_doublehook1 = DoubleHook()
PART_doublehook2 = DoubleHook()
PART_doublehook3 = DoubleHook()
PART_doublehook4 = DoubleHook()
ASSEMBLY_diaper_caddy.connect(PART_doublehook1.hook1, ENV_start.rod1)
ASSEMBLY_diaper_caddy.connect(PART_doublehook2.hook2, ENV_start.rod2)
ASSEMBLY_diaper_caddy.connect(PART_doublehook3.hook1, PART_doublehook1.hook2)
ASSEMBLY_diaper_caddy.connect(PART_doublehook4.hook1, PART_doublehook2.hook1)
ASSEMBLY_diaper_caddy.connect(ENV_end.hook2, PART_doublehook3.hook2)
ASSEMBLY_diaper_caddy.connect(ENV_end.hook1, PART_doublehook4.hook2)

end_frame = Frame([124.3,580,717.1], [-135.5,-40,20.5])
ASSEMBLY_diaper_caddy.end_with(ENV_end.hook2, end_frame)
```

A.4 FabHacks Solver Runtime

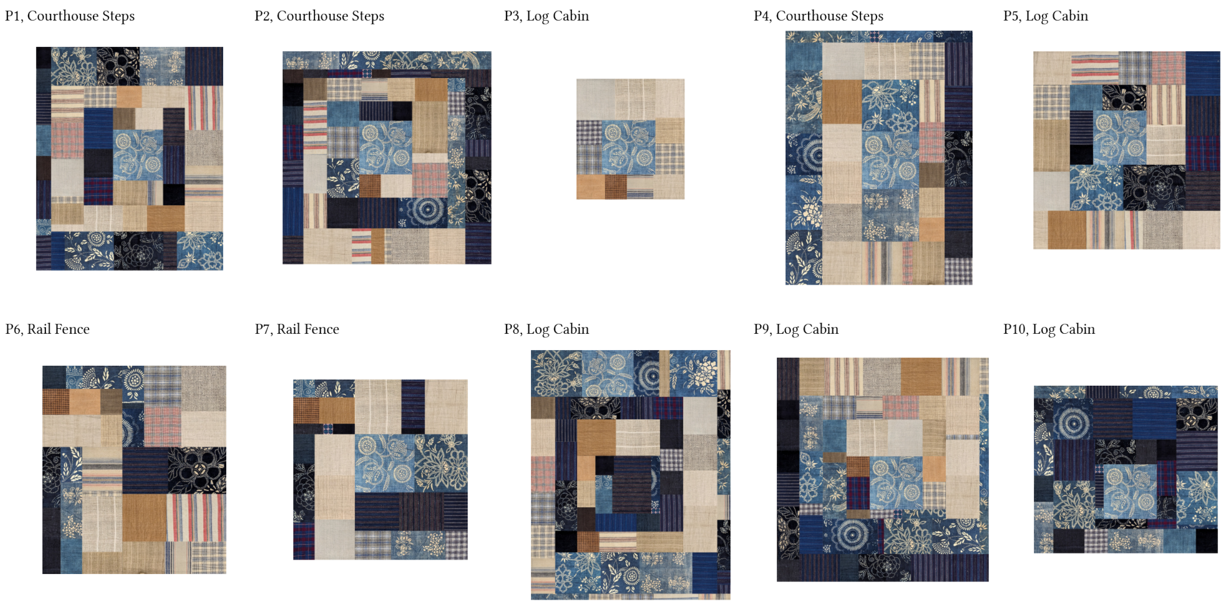
We used the example hacks as test cases and collected runtimes of various solver operations into Table A.5 below. The same laptop used for the user study sessions, a MacBook Pro (2020) with M1 Chip and 8GB memory, was used for collecting these data. In the context of UI interactions, step 3's "Run Optimization" corresponds to **Avg. Full Solve Time**, and step 2's pre-checks correspond to **Avg. Quick reject Time** and **Avg. Constraints Checking Time**. Based on the table, the runtime increases as the complexity (# Parts, # Cycles, # Params) of the example increases.

Table A.5: This table shows the runtime averages of solver operations. The first two columns are the example's name and the corresponding figure. The next three columns records the number of parts, the number of cycles, and the number of parameters (i.e., the degrees of freedom of the connections) in the hack design. The runtimes in the next four columns are collected as averages over 10 runs: (1) Avg. Full Solve Time: the average time taken for a full solve of the example; (2) Avg. Quick Reject Time: the average time taken for the *geometric quick reject* pre-check; (3) Avg. # Initial Guesses: the average number of initial guesses needed for checking constraints satisfaction of the example; (4) Avg. Constraints Checking Time: the average time taken for checking constraints satisfaction. The * next to the reading nook example means that due to its complexity, instead of using randomly generated initial guesses, the initial guesses are computed from optimizing individual chains in the design, which takes around 2-3 minutes.

Example	Figure	# Parts	# Cycles	# Params	Avg. Full Solve Time (sec)	Avg. Quick Reject Time (sec)	Avg. # Initial Guesses	Avg. Constraints Checking Time (sec)
demo	Fig. 4.24	4	0	10	9.997	-	-	-
toothbrush holder	Fig. 4.19a	3	2	20	15.551	0.003	1.0	2.027
charger holder	Fig. 4.19b	3	1	8	2.014	0.001	1.4	0.743
soap bottle holder	Fig. 4.19d	5	1	17	26.937	0.003	1.0	1.536
mug hanger	Fig. 4.19c	6	0	11	5.188	-	-	-
bird feeder	Fig. 4.15 left	10	1	16	45.734	-	-	-
paper towel holder	Fig. 4.19e	5	1	15	20.479	0.002	2.3	8.251
diaper caddy	Fig. 4.19f	6	1	18	50.839	0.002	1.3	8.227
bathroom basket	Fig. 4.22	4	1	14	28.151	-	-	-
clip lights 1	Fig. 4.26	11	0	25	30.820	-	-	-
clip lights 2	(from	11	0	25	28.293	-	-	-
clip lights 3	left to	9	0	20	19.182	-	-	-
clip lights 4	right)	9	0	20	18.685	-	-	-
reading nook*	Fig. 4.15 right	19	2	65	742.279	0.0146	3.1	336.741

A.5 ScrapPack User Designs

Figure A.3 shows the designs created by participants during the user evaluation.



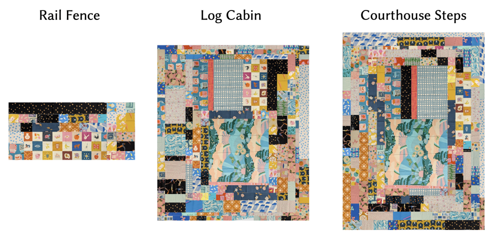
(a) Task 1 designs. To create contrast, most participants chose to bin their fabrics into a lighter bin and a darker bin. P3 preferred lighter fabrics while P10 preferred darker fabrics and focused on making sure the prints and solids patterns were in contrast. Several others tried to create spiraling structures of contrasting colors, or added darker strips as "boundaries" for the design. Most participants tried to make sure no two similar fabrics were next to each other.

(b) Task 2 designs. Compared to Task 1, participants all went for a different quilt block strategy. When designing the fabric for tote bag, participants all thought about how the fabric would look like made into a tote bag, and they tended to create more rectangular designs. Interestingly, P6 and P9 had in mind a bag pattern with a rectangular bottom that would be touching the ground, while P3 and P8 wanted to only create one side of the tote bag. P1 was the most cognizant of the finished size and tried to make it a size good for making a typical tote bag.

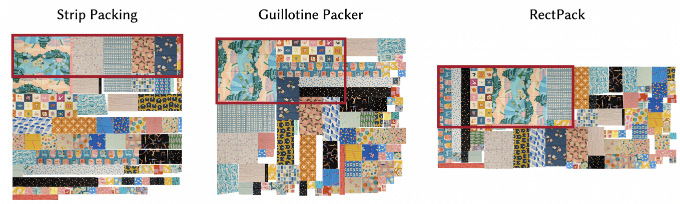
Figure A.3: Designs created by participants in Task 1 and Task 2.

A.6 Comparing ScrapPack with Automatic Bin Packing

Figures A.4, A.5 show additional visual comparisons between packing results generated via our method and those generated with automatic bin packing methods. The two fabric scrap sets here are the Etsy Cotton set and the Etsy Linen set.



(a) Our method with three packing strategies. The packing results' areas are as follows: Rail Fence - 481.15 in^2; Log Cabin - 2393.20 in^2; Courthouse Steps - 2311.16 in^2.



(b) Three automatic bin packing methods. The packing results' largest gap-free areas out of total bin areas are as follows: Strip Packing - 635.30 / 2809.52 in^2; Guillotine Packer - 785.25 / 2763.26 in^2; RectPack - 891.92 / 2670.20 in^2.

Figure A.4: Packing results comparison between ScrapPack strategies and three bin packing methods using Etsy Cotton set. The dark red rectangles indicate the largest gap-free rectangles within the bin packing results.

(a) Our method with three packing strategies. The packing results' areas are as follows: Rail Fence - 803.30 in^2; Log Cabin - 1369.71 in^2; Courthouse Steps - 1339.51 in^2.

(b) Three automatic bin packing methods. The packing results' largest gap-free areas out of total bin areas are as follows: Strip Packing - 294.67 / 1746.21 in^2; Guillotine Packer - 345.46 / 1691.63 in^2; RectPack - 365.62 / 1651.11 in^2.

Figure A.5: Packing results comparison between ScrapPack strategies and three bin packing methods using Etsy Linen set. The dark red rectangles indicate the largest gap-free rectangles within the bin packing results.