Tools for Digital/Physical Creativity

Hannah Twigg-Smith

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

Doctor of Philosophy

University of Washington
2024

Reading Committee:
Nadya Peek, Chair
Sayamindu Dasgupta
James McCann

Program Authorized to Offer Degree:
Human Centered Design and Engineering

©Copyright 2024
Hannah Twigg-Smith

University of Washington

Abstract

Tools for Digital/Physical Creativity

Hannah Twigg-Smith

Chair of the Supervisory Committee:
Nadya Peek
Human Centered Design and Engineering

Digital fabrication machines extend human creativity by enabling us to precisely manipulate the material world according to complex designs. Digital/physical creativity involves navigating relationships between digital design representations, physical processes, and emergent material behavior. Our interaction with this dynamic system is largely mediated by software tools, leading me to ask: How can software tools support digital/physical creativity?

This dissertation presents a series of four projects which I use to answer this question. To ground my work in current practice, I begin by studying an community of artist practitioners on #PlotterTwitter. I find that these artists rely on small, flexible tools that enable them to quickly iterate between digital designs and material outcomes. This insight informs development of three systems with which I contribute demonstrations of different strategies software tools can use to support digital/physical iteration.

First, I build Dynamic Toolchains, a live dataflow environment which demonstrates how extensible infrastructure can support end-user construction of feedback loops integrating diverse data representations, machines, and materials. Next, I build two versions of KnitScape, a design and simulation tool for knitting. The first shows how lightweight simulation can support development of intuition about emergent material behavior by enabling rapid exploration of unintuitive digital/physical relationships. The second shows how composable primitive elements enable this intuition to be applied to the creation of more complex designs. Together, these three systems serve

as examples of three kinds of digital/physical creativity support: exploration of workflows, materials, and designs.

More generally, each of these systems supports an approach to making I identify as *swatching*, a ubiquitous creative strategy in which small artifacts are made to reason about creative processes. I propose the term "swatchiness" to capture how well software tools facilitate small artifact iteration. I argue that swatchiness is a highly desirable quality in software tools which support digital/physical creativity, and I use insights gained from my empirical and systems work to synthesize and contribute four "Principles of Swatchiness" which developers can use to consider swatchiness in future tools: rapid insight, tinkerability, targeted support, and multimodal exploration. I argue that it is critical to build highly swatchy tools for digital/physical creativity because they will not only help us make more things today, but also help us imagine what we *could* make tomorrow.

Contents

1 Introduction **9**

Digital/Physical Creativity 10

Dissertation Overview 12

2 Tools, Tricks, and Hacks: Exploring Novel Digital Fabrication Workflows on #PlotterTwitter **16**

Introduction 17

Background 19

Machine Histories, Social Resistance, and Standardization 19

Craft and Computation 21

Online Creative Communities 24

Methods 25

Identifying Twitter Accounts 25

Background on Plotter Machines and Plotter Software 26

Details of #PlotterTwitter and Related Online Communities 26

Codebook 27

Data Collection 32

Coding and Memoing Process 32

Limitations 33

Findings 33

Data Overview 34

An Engaged Community Cultivates a Diverse Body of Knowledge 35

Foregrounding the Machine as a Co-creator 36

Lightweight and Flexible Design Tools are Preferred over a Monolithic Approach to CAD . . . 37
Grappling with the Digital/Physical Divide . . . 39
Niche Exploration Eclipses Generic Workflows . . . 41
Discussion . . . 43
Workflows Themselves Become the Medium . . . 43
#PlotterTwitter Defies Norms of Who Engineers and How . . . 44
Implications for HCI Research on Digital Fabrication . . . 44
Conclusion . . . 46
Acknowledgements . . . 46

3 Dynamic Toolchains: Software Infrastructure for Digital Fabrication Workflows 47
Introduction . . . 48
A Toolchain for Plotting Maps . . . 51
Related Work . . . 53
Computational Abstractions for Digital Fabrication . . . 53
Alternative Fabrication Interfaces . . . 55
System Design Goals . . . 56
Multiple Forms of Engagement . . . 57
Features Across the Digital/Physical Divide . . . 57
System Implementation . . . 58
Toolchain Dataflow Environment . . . 58
Module Development Framework . . . 60
Demonstrating Dynamic Toolchains . . . 64
Material Explorations: Interactive Watercolor Painting . . . 64
Connecting Diverse Inputs and Outputs: Embroidered Audio . . . 67
Multiple Levels of Abstraction: Milled Automata Panels . . . 68
Flexibility When Adapting Software: Automata Knits . . . 69

3D Toolpath Generation: Velocity Printing . . . 70
Integrating Libraries Across Languages: Plotted Maps . . . 70
Workshop . . . 71
Discussion . . . 73
Conclusion . . . 75

4 KnitScape I: Computational Design and Yarn-level Simulation of Slip and Tuck Colorwork Knitting Patterns 77

Introduction . . . 77
Background and Related Work . . . 79
Core concepts: Knitting with Slip and Tuck Stitches . . . 81
Existing Design Tools for Textiles . . . 83
Simulating and Rendering Fabrics . . . 86
Swatching with KnitScape . . . 87
System Implementation . . . 90
KnitScape Parameters . . . 90
KnitScape Swatch Simulation . . . 93
Example Outputs . . . 96
Swatch Knitting Procedure . . . 97
Changing the Stitch Pattern: Strategic Slip and Tuck Placement . . . 98
Planning the Color Sequence . . . 101
Removing needles from work . . . 103
Discussion . . . 107
Conclusion . . . 109

5 KnitScape II: Abstracting Knitting Design Elements with Blended Raster/Vector Primitives 110

Introduction . . . 110
Related Work . . . 115

Design Representations for Knit Objects . . . 115
Fabric Modeling, Simulation, and Rendering . . . 117
Rasterization . . . 117
Blended Primitives . . . 118
Pattern Elements . . . 119
Design Interface . . . 124
Example Workflow . . . 126
Demonstration Examples . . . 129
Cables, Braids, and Twists . . . 129
Fair Isle Colorwork . . . 130
Intarsia Colorwork . . . 132
Lace . . . 132
Further Applications: Sensors and Folds . . . 134
Origami-Inspired Pleat Patterns . . . 134
Intarsia-based Sensing Structures . . . 135
System Implementation . . . 138
Stitch and Yarn Chart Rasterization . . . 138
Yarn Separation and Transfer Planning . . . 141
Yarn Topology and Simulation . . . 142
Challenges and Opportunities . . . 144
Discussion . . . 145
Conclusion . . . 147

6 Swatchy Tools **149**

Swatchiness in Context . . . 150
Practical Swatching . . . 154
Principles of Swatchiness . . . 156
Rapid Insight . . . 157

Tinkerability . . . 159
Multi-modal exploration . . . 161
Focused Support . . . 163
Swatchy Mindsets . . . 164

7 Conclusion **167**
Future work . . . 167
Bottom-up meets top-down . . . 167
Swatchy tools for other domains of textile craft . . . 168
KnitScape (Out There) . . . 169
Reflection . . . 171

Appendices **213**

A Example Toolchain Module Implementations **214**
Axidraw Plotting Module Implementation . . . 214
Mapbox Module Implementation . . . 217

B Knitting Overview **220**
Glossary of terms . . . 221

C Pseudocode for Boundary and Path Rasterization **223**

Chapter 1

Introduction

During my orientation at Olin College of Engineering, every student received a big gray plastic toolbox. We were sent off to fill it with cheap logic analyzers, calipers, screwdrivers, and multimeters, and advised to write our names on our new tools in bold sharpie. Later, we were herded around to see the seemingly endless array of tools which were too big to fit in our toolboxes: 3D printers, sewing machines, vinyl cutters, laser cutters, hot wire cutters, drills, mills, lathes, and saws... We were told that with these tools we could make almost anything, and we nodded with wide eyes.

We received a digital toolbox to complement the physical one: a brand new laptop loaded with software like SolidWorks, MATLAB, COMSOL, and the entire Adobe suite. While spending the next few years learning how to use these tools to design and build physical things, I also learned that I love programming and building software. My senior capstone project at Olin was sponsored by SolidWorks, who prompted my team to build a "3D $\rightarrow$2D $\rightarrow$3D" design tool for kids (Figure 1.1). This project ended up being foundational to my future research interests and eventual path through graduate school. Through it, I realized I was not limited by the tools around me when I wanted to make something. I was fully capable of building my own.

Of course, I am far from the first human to realize she can build tools. We have been doing this for thousands–millions–of years. Our tools had a modest start: hunter-gatherers began by smashing rocks together in pursuit of a sharp edge, which they could use to hunt new animals, process food, and cut down trees. When available tools are insufficient for the task at hand, we are compelled to make new ones. Using tools to make things – and building *new* tools to make *new* things – is central to what it means to be human.

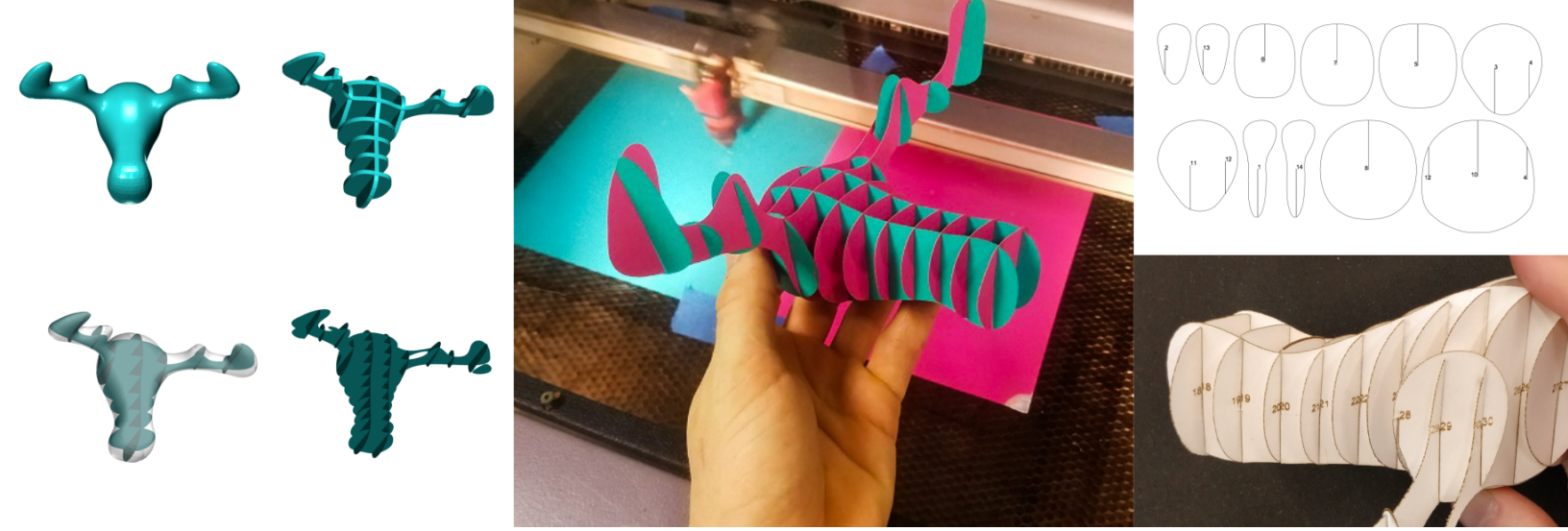


Figure 1.1 *My undergraduate capstone team at Olin College built Slice-it, a tool for generating customizable cut files for 3D models designed in the SolidWorks Apps for Kids suite. Here, a model of a moose head is sliced, cut from card stock with a laser cutter, and assembled by matching the slot labels.*

Digital/Physical Creativity

Creativity can be broadly defined as the capacity to create original works, ranging from ideas, discoveries, and solutions to artworks, compositions, and inventions. In his influential book studying the lives of nearly one hundred exceptionally creative people, Csíkszentmihályi emphasizes that creativity should not be seen as a trait which is internal to an individual person. Rather, it is a dynamic system shaped by deep engagement with a domain, a network of people, and individual characteristics [41].

Tools perform a crucial function within the broader system of creativity: they extend our natural capabilities and help us channel our creative will. While we direct our tools, they also direct our understanding of what we can (and cannot) create. As McCullough [153] writes in *Abstracting Craft*, "Choosing the right tool is not only about completing a task, but also about establishing and focusing that task. Holding a tool helps you inhabit a task."

This dissertation is broadly about the role tools play in creative workflows. I define a *workflow* as the series of steps someone takes to move from idea to outcome. Creative workflows involve a variety of tasks, abstractions, materials, systems, and of course: tools. I am specifically interested in how we might go about building *software* tools to support more aspects of *digital/physical creativity*: workflows in which digital tools are used to produce physical things.

Slice-it is a tool for digital/physical creativity: it supports a workflow where digital

3D models are turned into physical interlocking sculptures. There is some creative flexibility in workflows which use Slice-it. The input 3D models can be designed in a variety of programs or downloaded from the internet. The output cut files can be used to cut different materials in different ways: slices could be milled from plywood, laser cut from cardboard or card stock, or printed on paper and cut by hand. Despite this flexibility, using Slice-it establishes a commitment to the "3D →2D →3D" digital/physical workflow which was defined by my capstone team while building it.

Today, global access to computation means that we have more digital tools than ever which we can use to make things. Computer-aided design (CAD) software like SolidWorks, Fusion, and Onshape enable us to design intricate geometries, build complex assemblies of parts, and inspect how they might move together. Computer-controlled machines like 3D printers and laser cutters can manipulate material with precision that would be impossible by hand. Computer-aided manufacturing (CAM) software like slicers for 3D printers help us translate digital representations into programs that machines can understand.

Without tools like these, most digital/physical workflows would be prohibitively tedious and labor-intensive, if not outright impossible. By building new software and hardware tools, we move closer to a future where "making anything" is a reality [68]. However, access to tools is not enough. It takes a commitment to practice to learn to work with these tools, to use them to explore, and to understand how you can use them to shape the world. McCullough [153] continues:

> "You must learn how to bring skills and intentions together. You must learn how each tool works, how one tool works with another, and how all are maintained. You must know what tools are for. A well-equipped woodshop might make you feel like building something, but only if you are so inclined, and only if you are in practice."

This idea is central to my work. Beyond expanding the space of possible outcomes, I am interested how software can support other important aspects of digital/physical creativity: iterative exploration across the digital/physical divide, development of intuition about the relationship between material and process, and playful engagement with the activities of making. Therefore, the majority of my research, and thus the bulk

of my dissertation, has been devoted to answering the following research question:

How can software tools support digital/physical creativity?

My goal is to provide a series of recommendations which can inform future development of tools for digital/physical creativity. This dissertation describes four projects which help me develop these recommendations. I draw heavily on design principles for creativity support [200, 168, 219]. To answer my research question, I use empirical observations to inform development of software tools for digital/physical creativity. I began by studying an artist community using, building, and sharing tools to support their creative work. I use insights from this formative work to scope and build my own software tools, exploring how both general-purpose modular software infrastructure and highly-specific design tools for knitting can support digital/physical creativity. Together, this work comprises four papers published during my PhD.

Dissertation Overview

My dissertation presents four projects which address my research question. These projects provide a basis for formulating general recommendations for developers building tools to support digital/physical creativity. Each project demonstrates how tools can enable someone to engage with emergent behavior at the digital/physical divide via different levels of creative exploration: workflow, material, and design.

In Chapter 2, I begin by studying people to ground my work in current practice. I describe a formative study of workflows shared by artist practitioners under the #PlotterTwitter hashtag. This study provides a series of concrete examples which demonstrate how creative practitioners navigate the digital/physical divide. These artists pioneer increasingly novel digital/physical workflows by working within and beyond existing computational abstractions, often developing and sharing their own tools. Crucially, activity on #PlotterTwitter challenges the traditional CAD/CAM/CNC workflow that dominates digital fabrication. Artists demonstrate a clear preference for lightweight, flexible tools when navigating the digital/physical divide, and findings from #PlotterTwitter clearly show that creativity thrives when people have the agency

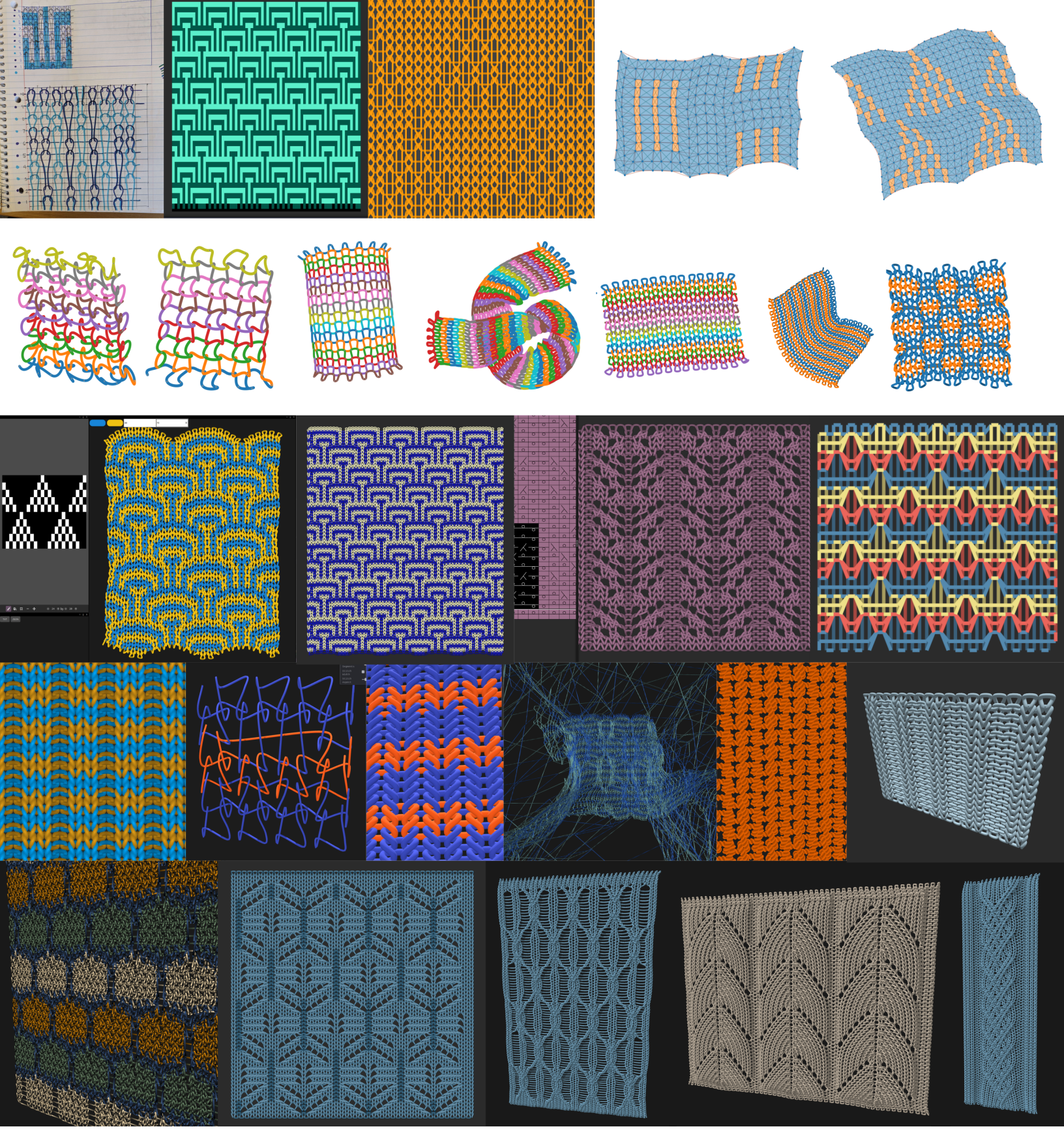


Figure 1.2 *Creative work is inherently iterative and nonlinear. The yarn simulation I developed for KnitScape (described in Chapters 4 and 5) evolved along with my goals and interests. I began using sketches and basic tilemaps (top row) to understand an unintuitive knit color patterning technique. Following this initial curiosity, I learned about knit fabric topology and build a 2D yarn simulation to better visualize these patterns (rows two and three). From there I continued on into three dimensions, using this project as a way to explore 3D rendering, shaders, and WebGL (rows four and five).*

to customize their processes.

In Chapter 3, I use insights from #PlotterTwitter to direct development of Dynamic Toolchains, an extensible dataflow environment for digital fabrication. Dynamic Toolchains is general infrastructure for "filling in the gaps" between existing software tools and libraries. Toolchains can serve as feedback loops between the digital and physical world, enabling rapid exploration of the emergent behavior of materials. Dynamic Toolchains demonstrates the importance of flexibility and modularity in tool design, showing how general infrastructure can support digital/physical creativity by enabling exploration *of* processes.

However, we can only get to a point where we are ready to build new processes after taking time to explore *within* processes. The later part of my dissertation narrows to investigate digital/physical creativity support in the domain of knitting, which provides a concrete reference point of within-process exploration. In Chapter 4, I identify slip and tuck colorwork as a concrete example of a material process which defies abstraction under traditional design representations. I build KnitScape, which supports rapid exploration of this colorwork process via yarn-level simulation. KnitScape provides a clear demonstration of how "swatching"–creating small sample artifacts to reason about the relationship between design and material– can be explicitly supported in software.

Systems can help us develop intuition, and they can also help us apply it to the creation of new designs by giving us composition tools. In Chapter 5, I extend the initial KnitScape system to create composable design primitives, demonstrating how they can be used to abstract a variety of knitting elements. The composable primitives editor enables someone to build custom abstractions as they notice patterns, giving them a way to explore knitting processes at multiple levels of abstraction. Together these two chapters provide a concrete example of a bottom-up approach to the development of software tools for digital/physical creativity. In building these tools, I was motivated by the same curiosity, personal enjoyment, and satisfaction I saw on #PlotterTwitter.

Taken together, these four projects provide concrete examples of how software tools can either support or hinder the creative process, specifically in workflows that span the digital/physical divide. They show how creativity can be supported at different levels: the workflow, material, and design. This work contributes to a broader

understanding of how to design tools that allow for playful, iterative exploration and help users develop intuition about material behaviors. The findings challenge the current paradigms of digital fabrication and propose new directions for building tools that are more responsive to the needs of creative practitioners across disciplines.

In Chapter 6, I synthesize insights from the four projects described above to develop design principles to guide development of future tools which support creative exploration across the digital/physical divide. I identify *swatching* as a ubiquitous approach to creative work through which we reason about processes by iteratively creating small artifacts. I propose the term *swatchiness* to capture how well a digital/physical creativity support tool facilitates swatching, and contribute four "Principles of Swatchiness" which can inform the development of future tools. I consider my own systems work under these four principles, and conclude with a discussion of how highly swatchy tools can encourage a "swatchy mindset" which I argue is a broadly useful strategy for creative activity.

Chapter 2

Tools, Tricks, and Hacks

Exploring Novel Digital Fabrication Workflows on #PlotterTwitter

In order to consider how software tools can support digital/physical creativity, it is critical to understand how existing digital fabrication infrastructure fits into the workflows of today's practitioners. This chapter explores the workflows shared by an online creative community of artist practitioners (#PlotterTwitter) who use drawing machines (plotters) to create artworks. We focus on how and why practice diverges from envisioned use cases, and what that might mean for developers of future tools.

The workflows shared on #PlotterTwitter emphasize the highly contextual and iterative nature of digital/physical creativity. Artifacts evolve as artists probe relationships between digital designs, machine precision, and physical materials. Harnessing emergent behavior at the digital/physical divide is a primary form of creative exploration on #PlotterTwitter. This is at odds with the dominant CAD/CAM/CNC paradigm, which largely minimizes opportunities for emergence. Artists clearly prioritize tools which enable fine control over low-level interactions and quick iteration across the digital/physical divide.

The content of this chapter was adapted from a paper published at CHI 2021 [244]. The insights I gained from studying #PlotterTwitter have proved foundational to my systems development work, including the systems covered in later chapters.

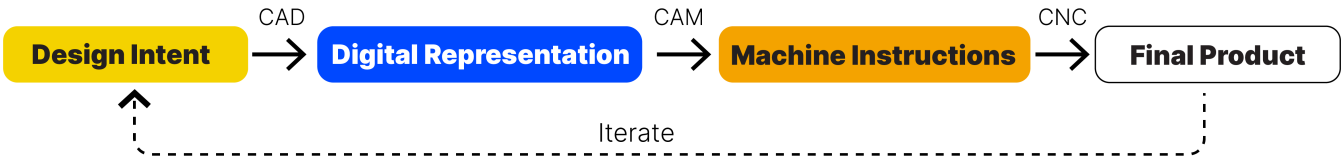


Figure 2.1 *The canonical digital fabrication workflow implies that work follows a linear progression through CAD, CAM, and CNC.*

Introduction

As computer-controlled digital fabrication machines become more affordable, they become increasingly accessible to a diverse user base. Online hobbyist communities have cropped up around digital fabrication machines, comprising people who use and customize both the machines and their supporting software tools [176, 10, 214]. These organic communities, made up of people with various backgrounds and levels of expertise, also create and share bespoke digital fabrication *workflows*. A workflow is series of tasks someone undertakes to move from idea to outcome, which can involve a variety of tools, techniques, data representations, materials, machines, and systems.

Much of the software and hardware infrastructure underlying digital fabrication tools today was developed for engineering settings in the mid-twentieth century. This infrastructure prioritizes standardization and repeatability for expert users, which has led to a system of normative ideas of what constitutes engineering as distinct from art and craft. Over time, this has resulted in a prescriptive stance of *how* tools for digital fabrication should be used even beyond traditional engineering environments, replete with assumptions about who the users of the technology might be.

In contrast to traditional engineering environments, *personal* digital fabrication machines allow for highly customized, small-scale manufacturing. This chapter explores an online community, **#PlotterTwitter**, which challenges normative descriptions of digital fabrication. Instead of exclusively working within existing prescriptive workflows, #PlotterTwitter artists claim the agency to develop personal, ad hoc software tools that support their individual artistic goals and interests. By critically examining where individuals are creating their own software and hardware tools, we can

understand how makers in online communities are pushing the boundaries of what can be produced with these tools.

Many HCI researchers and practitioners continue to theorize the whole of digital fabrication as a *canonical workflow* that proceeds rigidly through several consecutive stages: a design intent, a digital representation, machine instructions, and a final product, as schematically shown in Figure 2.1. This is what is heralded as the future in which "data becomes things" [68]. However, the data we gathered from online fabrication communities show us that the range of workflows that members are pioneering extends far beyond what can be captured in this common-case canonical workflow. As previous "edge cases" become the new status quo, it is more important than ever to develop models of digital fabrication workflows that account for people who are building their own infrastructure for digital/physical creativity.

To examine how online digital fabrication communities are working beyond the canonical workflow, we conducted a qualitative case study of #PlotterTwitter, a community of Twitter users who document and publish their artistic work under the eponymous hashtag. Members of #PlotterTwitter use *plotters*, computer-controlled drawing machines that move a pen or other writing instrument to produce a 2D drawing. Members share final drawings that result from their workflows, but also share works-in-progress, screenshots and code from software tools, photos of machine use and machine failures, and snippets from bespoke workflows-in-progress.

We chose to study #PlotterTwitter in particular due to the low barrier to entry to plotters compared to that of other digital fabrication machines. Furthermore, participants arrive at #PlotterTwitter from a wide range of backgrounds outside of maker communities, such as craft and illustration. As a result, #PlotterTwitter offers a diverse sample of community members of varying levels of expertise who document their activities frequently and publicly. While the medium of Twitter introduces some constraints, such as the brevity of tweets compared to longer-format tutorials or forums, #PlotterTwitter documents an evolving collection of workflows with both a low barrier to machine use and a low barrier to posting content and engaging in discussion.

Given our goal of better understanding the diverse and social development of plotter workflows, we propose the following research questions:

1. How can we conceptualize the space of workflows on #PlotterTwitter beyond the canonical workflow?
2. How does work from #PlotterTwitter question assumptions of who uses digital fabrication tools, and how?

To address these questions, we conducted an analysis of 1500 tweets from #PlotterTwitter that were collected over five months in 2020. To guide our analysis, we created a set of codes corresponding to all possible transitions between steps in the canonical workflow. These codes are not intended to fully capture the range of workflows on #PlotterTwitter—instead, they represent the extent to which we can conceptualize workflows when using the canonical workflow as a theoretical frame. They also give us a starting point for discussing the niche and personalized processes of personal digital fabrication. To this end, while coding tweets, we paid particular attention to activities that were were captured poorly (or not at all) by the canonical workflow. We created memos of serendipitous observations and used them to identify and develop themes. We detail and discuss each theme, and use them to argue for a more inclusive understanding of the breadth of workflows in digital fabrication.

Background

Machine Histories, Social Resistance, and Standardization

Demands for metal products during the Second World War intensified development of new machines and technology for making these products. The invention of computer-numerically-controlled (CNC) machinery was driven by a need for efficiently produced complex parts. However, beyond efficiency, developing CNC also provided an opportunity to reshape power relations between machinist labor and management [174]. The technology that grew into CAD/CAM (computer-aided design/computer-aided manufacturing) was partially shaped by a desire to reduce reliance on skilled labor. Stakeholders at the time argued that the separation of programming CNC machines (complex, happens once) from running CNC machines (once coded, can easily be repeated) could facilitate deskilling manufacturing and reducing production cost. Production could scale up with an investment in machines, rather than people.

This separation in programming and running machines persists in contemporary digital fabrication in a canonical workflow where steps can be categorized as CAD - CAM - CNC (see Figure 2.1). These steps separate an object's design, its manufacturing plan, and its manufacturing process. This categorization of steps is familiar to the HCI community—for example, in describing the stages newcomers go through when 3D printing, Hudson et al. name *1. Plan, 2. Model, 3. Setup, 4. Verify, 5. Print* [90]. They found that the expertise required for 3D modeling in CAD was a major barrier to newcomers, and thus many novices trying out 3D printing turned to websites such as Thingiverse in search of pre-made designs—substituting downloading for modeling. However, Hudson et al. [90], a more in-depth analysis of activity on Thingiverse [10], and a study of digital fabrication practitioners [262] show that design decisions made in CAD are tightly coupled to viable choices in CAM and CNC. They emphasize that successful digital fabrication relies on contextual information, design clarification, and expert tips that inform other stages of making.

In *Forces of Production*, Noble, through extensive anthropological research, describes how although companies adopting CNC equipment in the 1960s argued the CNC was easier to operate and therefore operators should be compensated at lower rates, labor tasked with operating the equipment found its operation to be error-prone, to require greater attention and anticipation to avoid collisions and problems, and to cause more worker tension and fatigue. To effectively run the machines, workers relied on skill and experience to know when to pause operation, make adjustments, or check tolerances. Protracted conflict with labor eventually resulted in companies conceding that operating CNC equipment needed to be done at a rank equal to if not higher than those operating conventional equipment [174, Chapter 11]. HCI and STS scholars have a long-standing tradition of studying such disconnects between representations of work and practices of work including in laboratory studies [126, 121, 207], entrepreneurship [142, 101, 74], and medicine [161, 54]. We argue that digital fabrication workflows would benefit from further scrutiny distinguishing between promise and practice, and position our contribution in this tradition.

HCI fabrication research contains many systems contributions, for example providing proof-of-concept implementations of novel workflows [269, 205]. Many of these contributions are not made for existing communities of practice (such as machinists or craftspeople), but for "novices", "consumers", or "non-engineers" who are predicted

to engage more with digital fabrication in the near future if related technology is made increasingly available and accessible. To underline how non-experts could engage with fabrication through these systems, many are described as “end-to-end”, meaning that the system encompasses all steps seamlessly and no further workflow development or problem solving is required from the user.

Examples of end-to-end fabrication systems for novices include those for designing and making laser-cut objects [16], pneumatic devices [149], bottle-based assemblies [123], laminated mechanisms [136], knit objects [169], button pressing systems [197], origami robots [155], or molds for casting [247]. Each of these systems offers a clear pathway for design and fabrication within a particular workflow, but not necessarily a clear picture for who might adopt those workflows and why.

Suchman contends that human actions are always situated and contextual [229], which erodes the value of pre-determined plans. We believe this erosion applies to many seamless fabrication systems. On #PlotterTwitter, we observe extensive resistance to the prescriptive plan of the canonical workflow. This is despite the breadth of backgrounds, novice or otherwise, present in the #PlotterTwitter community. We argue that the categories of the canonical workflow are too constrictive to describe #PlotterTwitter’s digital fabrication practice. Furthermore, these categories have origins in a system designed for control, rather than agency. We identify a need to understand how people practice digital fabrication in context, such that we can approach the design of digital fabrication technology while keeping human agency in the center of our focus.

Craft and Computation

Scholars have already explored the intersection of craft and computation through various angles. We are most interested in expanding on work that asks: what happens when machines become co-creators of craft, rather than static automators of pre-determined plans? Andersen and colleagues examine this explicitly through their notion of *digital crafts-machine-ship*, writing that “in this dialogue of closeness and distance, we are still making and changing the machine, and what the machine makes. This is live coding or live building: we touch the system and the system touches us

back" [11].

The idea that the maker is affected by what they are working with—from materials, to machines and tools, and now to the software that controls machines—is what social anthropologist Ingold terms *morphogenetic design* [98]. Morphogenetic design contrasts with *hylomorphic design*, inherent to the canonical workflow, which positions the maker as the single source of design intent and machines as fixed implementors. Ingold writes that "whenever we read that in the making of artifacts, practitioners impose forms internal to the mind onto a material world 'out there,' hylomorphism is at work." Driven to imagine digital fabrication outside of hylomorphism, Devendorf et al. propose the notion of *post-anthropocentric* making where the maker "becomes innovative and technologically fluent by working closely with machines and materials, listening and sensing their actions, and responding accordingly" [47]. They and other scholars have proposed systems that highlight the agency of the material, from the the materiality of light [216, 237], to hybrid smocking [53].

Of particular note is *FreeD*, a hybrid carving tool that incorporates both the manual action of the maker alongside the data from a 3D model [269]. More recently, Albaugh et al. built an "undeterministic" loom where the final result is influenced by the actions of the weaver, sampled camera imagery, and community participation via the Twitch streaming platform [6]. While these works show curated instances of morphogenetic crafting with machines, we are interested in how morphogenetic craft is happening within #PlotterTwitter. We seek to understand where makers at a grassroots level organically depart from hylomorphic notions of digital fabrication.

Conceptions of the machine as a co-creator led us to explore the relationship between #PlotterTwitter artists and their self-made software tools. Software tools for generating vector graphics, particularly prominent on #PlotterTwitter, are ephemeral and distinct from the physical tools that craft literature has often focused on. They occupy an intermediary place in the creators workflow, generating a plan for what is to be drawn without drawing it. These tools, and the abstractions required to create them, have the effect of separating the creator from their materials [104]. Of physical ad hoc tools in hackerspaces, Bardzell et al. write: *"The tool not only clarifies one's work but also situates one in relation to the world, understood in both physical and also social senses."* [15]. Wakkary and Maestri further reinforce this idea through

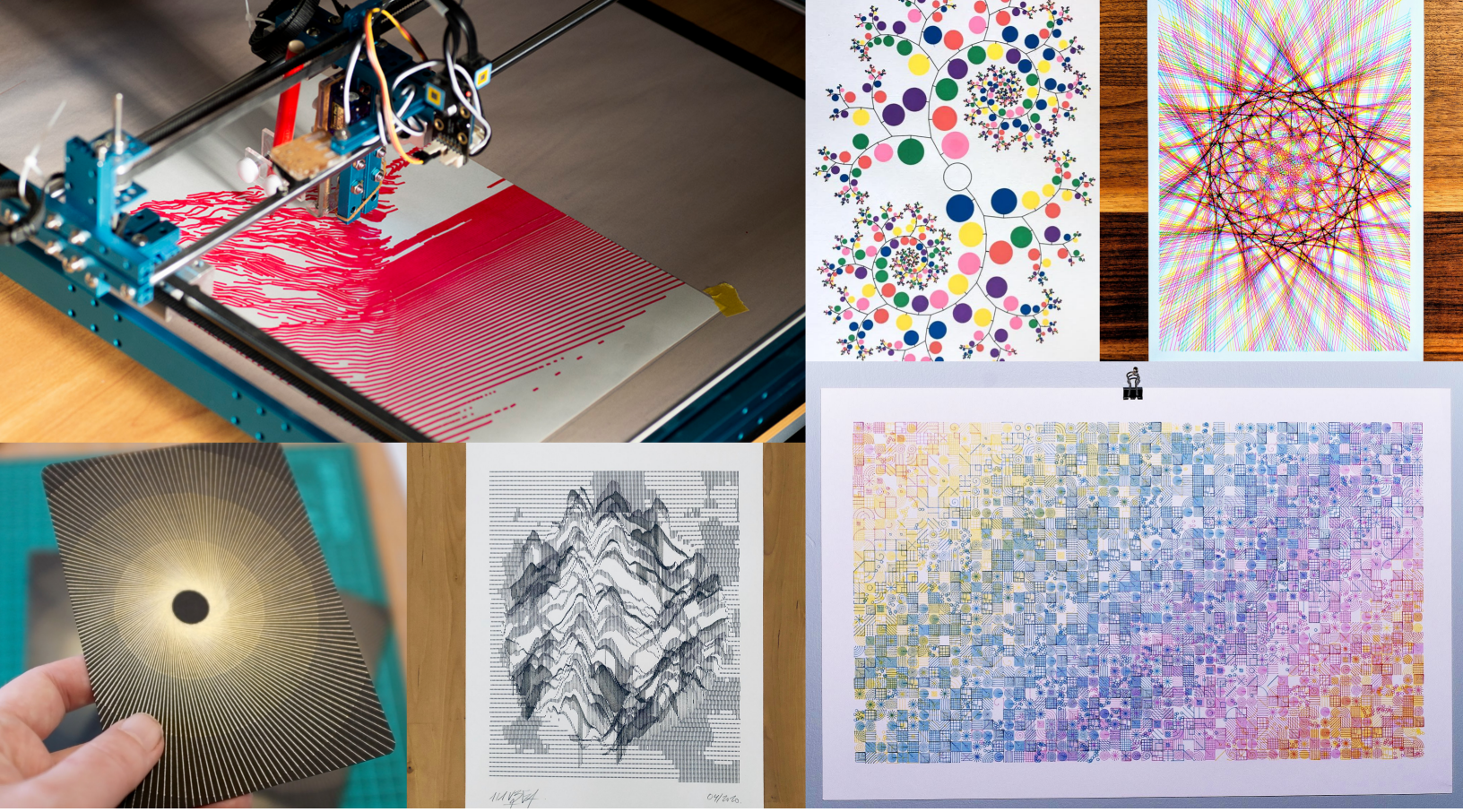


Figure 2.2 *#PlotterTwitter images shared by @NadiehBremer, @v3ga, @jcupitt65, @paulrickards, @revdancatt, and @kenjirototsuji.*

their ethnographic study of resourceful design in the home. [253] Now, how might this sentiment evolve when applied to software tools for digital/physical creativity?

We also draw inspiration from HCI scholars who draw from theories of craft more generally. Blauvelt et al. examine how computational craft re-informs interpretations of hardware and software [18], and Rosner calls attention to the role of craft within the development of computing technology, and its implications for the field of design [209]. Frankjaer and Dalsgaard contribute a review of craft-based inquiry in HCI leveraging Sennett's tripartite model of craft [217, 65]. We can observe their notions of "creating knowledge through deep, embodied engagement" in the work of various artists working at the edges of digital fabrication [202, 210, 157]. Scholars have leveraged paper in particular, from circuit stickers [195], animation [194], kirigami input devices [266, 34], foldable mechatronics [177, 215], to machines themselves [185]. Other researchers have explored ways of bringing craft-based thinking into computational mediums, notably, Jacobs's Dynamic Brushes [105], He and Adar's embroidering plotter [81], and Tian et al.'s improvisational woodworking tool [231].

Prior work has also examined how digital fabrication tools are gaining traction in settings where they have not previously existed. Quitmeyer and Perner-Wilson studied digital craft in harsh environments [196], while Jacobs and Zoran explored combining the traditional crafting techniques of Kalahari hunter-gatherers with computational tools [103]. Saegusa et al. and Devendorf et al. have explored how digital fabrication tools disrupt or integrate with traditional craft workshop settings in a pottery studio and a weaving workshop, respectively [212, 48].

Overall, the coupling of digital fabrication and craft is an active area, and researchers have investigated its intersection across many domains. While relevant to this body of research, our work differs in that we are focusing on studying *makers* who are are engaging with craft and digital fabrication in online communities. We investigate how people are not only redefining craft through computation, but redefining computation as part of their craft on #PlotterTwitter.

Online Creative Communities

Prior scholarship of online creative communities has proliferated, covering a vast range of domains. We are most interested in work that explores themes found in #PlotterTwitter such as do-it-yourself (DIY) ethos, peer production, and remix culture. One thread examines the Scratch programming language and its associated community, which has expanded far beyond its initial goal of promoting children's learning. Studies of this community examine trends in creative collaboration, [208], remixing [84], and attribution [162]. Programming languages dedicated to creative coding, such as Processing, p5.js, and openFrameworks, have enabled people to create art with code. Communities emerge from each language's respective forum [193, 179], often contributing back to the design of the language [181].

More specifically, researchers have studied how entire processes are documented and shared across the internet. Kusnetsov and Paulos highlight online DIY communities such as Instructables, Adafruit, and Etsy [124], finding that such communities emphasize open sharing and learning over profit and social capital. Tseng and Resnick study Instructables in-depth, analyzing how creators document their projects [240]. Other scholars have studied online fan fiction communities [12, 56]. In particular, Kim

et al. contribute a prototype tool to allow small online groups to crowdsource the writing of short stories [119]. Some of these online communities, such as the World of Warcraft machinima community, develop their own tools in tandem with establishing a community aesthetic and visual skill [182]. Digital fabrication also has its share of online platforms that enable model remixing [176] which have inspired novel systems that allow creatively reconfiguring mesh files [236] and linking 3D objects [86]. Our interest lies in identifying how lessons from prior work carry over to #PlotterTwitter, where configuring the underlying software and machine infrastructure becomes part of the craft itself.

Methods

During ongoing exploration of online maker communities, we identified #PlotterTwitter as a place where examples of diverse practice are shared and iterated upon by a wide audience. We are interested in documenting the variety of techniques people use to create their plotted artwork. To analyze workflows on #PlotterTwitter, we developed a coding scheme to describe workflow steps. We collected data from #PlotterTwitter using Twitter's streaming API, and three coders assigned codes to the data using a custom browser-based tool. We combined the coded data and documented emergent themes, including where our new coding scheme failed to capture the nuances of individual workflows.

Identifying Twitter Accounts

We have decided not to anonymize the Twitter handles of the #PlotterTwitter artists whose works are included in this chapter. We have contacted or attempted to contact all 29 accounts that appear. All those who responded (25/29) have asked for their Twitter handles to be included alongside their work, mentioning that they use the hashtag for recognition and exposure. Given a clear community preference for named attribution, we believe it is still appropriate to credit the remaining four Twitter accounts for their creative work, even though we were not able to secure their explicit consent (see [27]). The four accounts we were unable to contact were seemingly anonymous

and not linked to identifying information such as human names, personal websites, or other social media.

Background on Plotter Machines and Plotter Software

Plotters are CNC machines that draw lines based on vector graphics. They were one of the earliest output devices available to computers, first appearing in the 1950s. Historically, they also had an advantage over conventional printers for producing large technical drawings, and were widely used in architecture and engineering. Vector graphics (e.g. SVG files) describe shapes as mathematical functions rather than by listing values of individual pixels. These functions include a direction, start and end points, and are rendered on-screen as a line or smooth curve.

However, plotters also deal with factors not captured by geometric abstractions, such as speed, direction, acceleration, and vertical tool movement. This information is invisible when viewing a vector drawing alone on a screen, but these variables can dramatically impact the resulting plot when paired with the physical properties of paper, ink, and other materials. This can require extensive tuning and iteration, making the translation from screen to physical product far from "push-to-print." For example, chosen speeds will produce lines of different thicknesses depending on the "ooziness" of a pen, and the direction of motion can produce significant differences in aesthetics when using a flexible tool like a paintbrush.

Plotters are no longer widely used in industry, but they have been adopted by craft and hacker communities. Popular brands include Axidraw, Cricut, and Silhouette. To convert from vector file formats to plotter motion, the vendors provide control software, similar to a printer driver, that assists the users with creating a toolpath.

Details of #PlotterTwitter and Related Online Communities

The #PlotterTwitter hashtag is actively used on social media platforms including both Twitter and Instagram to share content relating to the use of plotters, usually for the creation of art. Figures 2.2 and 2.6 contain example images from our dataset that we consider to be representative of work shared on #PlotterTwitter. Other communities

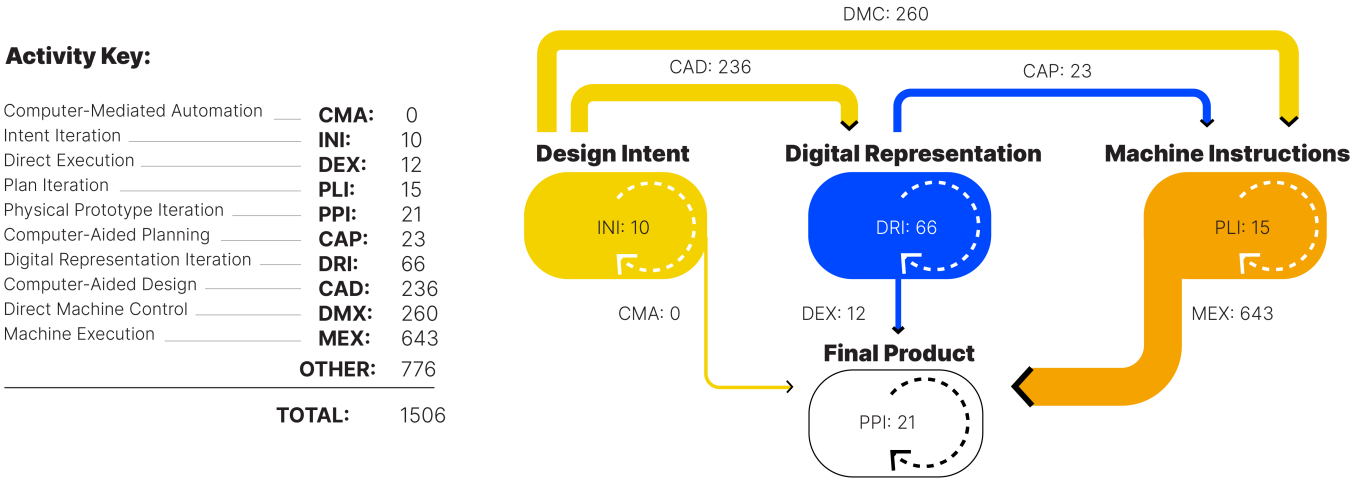


Figure 2.3 *A list of our codes and their occurrences. Our codes draw additional connections between categories of the canonical workflow, describing practices that cannot be directly categorized as CAD, CAM, or CNC.*

devoted to drawing machine discussion have emerged outside of Twitter, including dedicated groups on Facebook, Discord, and Reddit. The most active of these other communities seems to be the DrawingBots Discord server, which had approximately 800 members in January 2021 and has since grown to over 2500 members in November 2024.

In addition to sharing completed and in-progress works, members of these communities ask questions, share resources and ideas, and engage in other general discussion related to plotters and drawing machines. In early 2019, a member of the community created `www.drawingbots.net`, a website for compiling resources for plotting, such as vector graphics generators, pen and paper recommendations, and numerous custom software tools for eliciting unconventional behavior from their plotters. Since 2018, the Discord community has organized a number of in-person meetups and artwork swaps in London, New York City, and San Francisco.

Codebook

Through initial grounded observations, we noticed that the canonical CAD/CAM/CNC workflow often used to frame HCI fabrication research was not able to capture the breadth of #PlotterTwitter workflows. We found that artists often shared tools that

they had made themselves, or described activities that could not be accurately described as CAD, CAM, or CNC. Using the canonical workflow as a starting point, we connected the nodes of the workflow graph in Figure 2.1, naming additional edges between categories of the canonical workflow, describing practices that cannot be directly categorized as CAD, CAM, or CNC. We arrived at ten total codes for describing workflow steps, which are shown in Figure 2.3. We name and describe each workflow step code, and provide example tweets.

CAD: Computer-Aided Design

The term "CAD" is often used in engineering circles to refer to design software, often (but not exclusively) used for defining geometry. We are using "CAD" to describe any task that translates a user's design intent to a digital representation. Some examples of CAD software include 3D modeling programs like Blender and Rhino3D, and 2D vector graphics programs like Inkscape and Adobe Illustrator. For our analysis we also include self-developed tools in this category, like ones often used in creative computation such as Processing scripts or browser-based SVG generators. In an example CAD tweet, Twitter user @clarkkampfe shares a recent CAD tool they created (Figure 2.4): *"My latest. You can make your own with the tool I recently published at https://rasterize.live #plottertwitter Make sure to select the option to output SVG."*

CAP: Computer Aided Planning

This workflow step fulfills a similar place as the task colloquially known as CAM (Computer-aided Machining/Manufacturing). However, we opted to use the more generic term "planning" to better describe this step, as automation is not limited to machining or manufacturing. During the CAP step, a digital representation of a task or object is converted into machine instructions (in a language such as G-code) that can be executed on a machine. Software that provides this functionality includes Cura (for 3D-printing), or homebrew SVG-to-G-code conversion tools. In an example CAP tweet, Twitter user @lwrkalia asks for advice on converting SVGs to G-code: *"What do #plottertwitter people use for converting SVGs to g-code? Looking for something that can be automated."* After receiving some responses, they find a process that works: *"Right now I take my SVGs, then process with vpype then generate gcode with juicy gcode. Very happy with the results."*

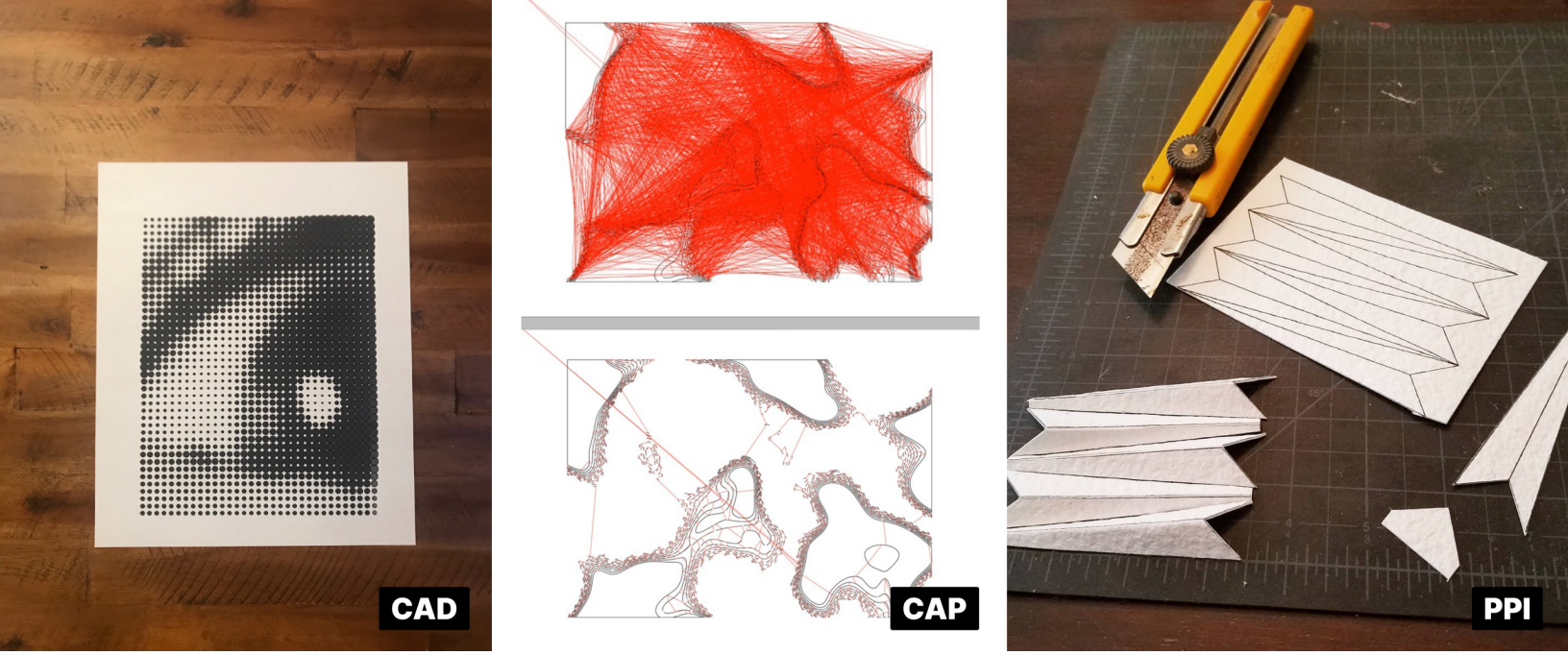


Figure 2.4 *The #PlotterTwitter community shares their work at many points in their processes, not only the finished product.* ***CAD:*** *A drawing made by @clarkkampfe with a custom CAD tool.* ***CAP:*** *Top: Unoptimized toolpath (pen-up movement in red). Bottom: Optimized toolpath. Via @thomasp85.* ***PPI:*** *@265design cuts and folds a plot.*

MEX: Machine Execution

During machine execution, a set of low-level machine instructions, such as G-code, is executed by a physical machine. In one tweet, @ManiacalLabs shares a video of their plotter plotting its own schematic, revealing G-code instructions that were generated with Lightburn, a program for controlling laser cutters.

DEX: Direct Execution

During direct execution, a separate set of machine instructions is not explicitly created. Instead, the machine is controlled directly from the digital representation. For example, the Axidraw Control plugin for Inkscape allows a user to directly control the Axidraw without exporting an SVG and converting it to G-code. An example DEX tweet by @mhkingsbury uses this plugin to draw a map (Figure 2.5): *"I used @anvaka's City Roads to build the map of @stlouispark and sent it to the AxiDraw for printing. Really happy with how it turned out. #PlotterTwitter."*

CMA: Computer-Mediated Automation

Computer-mediated automation includes systems where a machine immediately translates a user's physical input into a physical result. This code is similar to DEX, however,

instead of a completed set of machine instructions being run on a machine, individual commands are sent to the machine as a user makes decisions. While we did not find any examples of CMA in our dataset, systems dubbed as *interactive fabrication* [256] in HCI literature, such as Tian et al.'s Turn-by-Wire, would be considered computer-mediated automation. [232].

INI: Intent Iteration

Intent iteration represents activities that help refine the user's intent. This could include brainstorming, material explorations, sketches, or other processes. Intent iteration aids in solidifying future workflow steps by forming a more-complete end goal.

DRI: Digital Representation Iteration

Digital representation iteration describes iterating on existing digital representations. For example, someone could generate artwork in Processing, export it as an SVG, and use Inkscape tools to remove curves that they do not want to appear in the final artwork. In an example DRI tweet, @fastness describes an algorithm they created to convert digital photos to vector art: *"First draft of a photo to line drawing algorithm I've been thinking about for years, on and off. The bits where it goes wrong are sort of interesting at least... #plottertwitter."*

PLI: Plan Iteration

Plan iteration describes tasks that iterate on or directly modify a set of machine instructions. This could include compilation, path optimization, or validation. In an example PLI tweet, @thomasp85 shows the results of their path optimization tool (Figure 2.4): *"My #axidraw graphic device now has automatic travel optimisation as well as a preview rendering mode. Red lines are air travel #plottertwitter #rstats."*

PPI: Physical Product Iteration

This workflow step describes activities which take place by a human, often after a machine has been used. This includes tasks such as sanding, painting, or other finishing techniques. PPI could also describe human tasks that take place in parallel with the machine execution. In an example PPI tweet shown in Figure 2.4, @265design shares

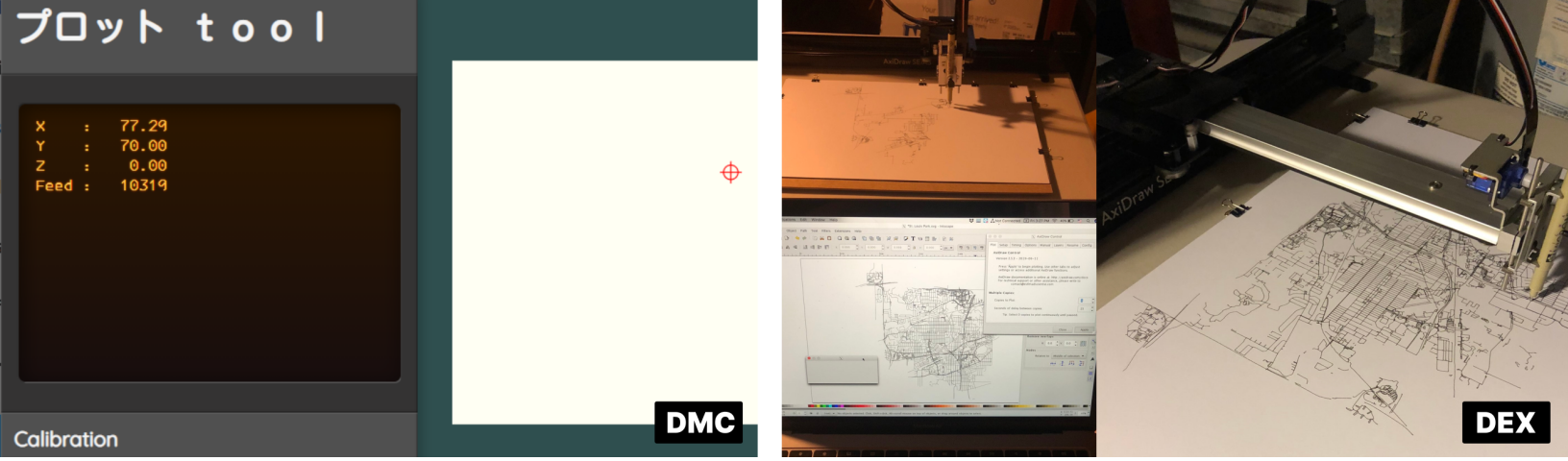


Figure 2.5 *In addition to artworks, #PlotterTwitter artists shared the tools they use to calibrate and control their plotters.* ***DMC:*** *A plotter calibration tool by @plotslol.* ***DEX:*** *@mhkingsbury controls an Axidraw with the Axidraw Control plugin for Inkscape.*

a workflow where they use a plotted design to cut and fold paper by hand: *"Digital to 2D to 3D #paperart #paperfolding #penplotter #plottertwitter #geometricart."*

DMC: Direct Machine Control

The direct machine control workflow step represents activities that bypass the creation of a digital representation. This could include manually writing lines of G-code or using machines that allow for record-and-playback functionality. We also consider machine setup activities such as zeroing or calibrating pen heights to fall into the DMC category, as the user generally directly enters commands to move the machine. A DMC example tweet by @plotslol demonstrates a calibration tool they created for their plotter Figure 2.5): *"my hobby isn't making art with a plotter, it's making artful calibration tools FOR my plotter #PlotterTwitter."*

OTHER

We use the "OTHER" code to describe tweets that could not be described the previously discussed workflow codes. This included tweets that were off-topic or participating in *#PlotterTwitter* meta-discussion: *"I discovered #PlotterTwitter today and my life has been changed" - @mimigenot* However, the "OTHER" code also allowed us to see where our coding scheme fell short of describing all of the activities on #PlotterTwitter. We found that many seemingly important workflow steps ended up included in the OTHER category, as they didn't fit with any of the previous categories. We took note of these additional workflow steps because they could guide development of a future workflow framework. One such example was when @liviopacifico used a 3D scanner to transfer real-world geometry to digital form to create a layered ribbon

portrait of their sister (Figure 2.6, right).

Data Collection

Our dataset consisted of 1500 tweets collected between 11 March 2020 and 9 July 2020. We used the Twitter Streaming API to collect tweets over this period, filtering them based on the *#PlotterTwitter* hashtag. We ran two separate instances of the streaming API on Heroku virtual servers to prevent possible data loss. Once the data had been collected, we ensured each tweet was represented only once by removing retweets, duplicates of the same tweet ID, and tweets from Plotter Bot (@Plotter-Tweeter), a bot that shares #PlotterTwitter content.

Coding and Memoing Process

Three coders individually assigned codes to the entirety of the initial dataset and the results were combined such that one tweet would receive all codes that any coder assigned to it. Because tweets could mention multiple steps of a workflow, they could be assigned more than one code. We developed a web application to allow us to efficiently process the tweets. The app loads a JSON file of Twitter data and displays the tweet text and images alongside the code options. The coders could quickly navigate the dataset and assign codes via the keyboard.

The coders set aside any tweets that were coded as OTHER that they felt were *serendipitous* discoveries. Rivoal and Salazar define serendipity as the "convenient notion to point out the very experience of suddenly coming to 'see' something that had previously been out of sight: the striking realisation that 'this is what it is about' [which] ethnographic narratives refer to as the 'Geertzian moment"' [206, 67]. The OTHER code intentionally represents both unrelated and non-descriptive tweets, but also highly descriptive tweets that could not fit any of our codes derived from the canonical workflow. The latter revealed unexpected "happy accidents" such as exploiting the inconsistencies in a plotter, or creating from scratch the missing "æ" glyph for a plotting typeface. The coders and authors collectively memoed and discussed these tweets, and the resulting emergent themes inform the majority of our findings.



Figure 2.6 *While many plots shared on #PlotterTwitter are abstract, #PlotterTwitter plots can be created from existing images or objects, such as photographs, paintings, and 3D scans. These plots were made by @PhotoPuck, @tasty_plots, @ajpiano, @yaronski, and @liviopacifico.*

Limitations

Interest in drawing machines extends beyond the *#PlotterTwitter* hashtag. Discord, Reddit, Facebook, Instagram, and other online forums host communities adjacent to #PlotterTwitter. Interviews with members of these communities could shed light on individual workflow patterns, as the 280-character limit imposed on tweets at the time of our data collection may prevent *#PlotterTwitter* users from sharing nuanced information about their workflows. By coincidence, data collection for this project began around the same time as stay-at-home orders prompted by the COVID-19 pandemic were imposed around the world. We did not compare our data to a pre-pandemic dataset.

Findings

The workflows displayed on #PlotterTwitter are incredibly diverse. Artists create their own tools for generating graphics and controlling their machines, and fluidly adopt existing tools as well. In this section, we provide a broad overview of the #PlotterTwitter content and describe the themes we observed in our dataset. We would like

to emphasize we are unable to make claims about *individual* practice based on the tweets we have analyzed, as we are essentially seeing out-of-context snapshots of individual workflows. Rather, and more to the point of our argument, the data shows the community's diversity by demonstrating the sheer breadth of practice that takes place on #PlotterTwitter.

Data Overview

The #PlotterTwitter hashtag encompasses a range of content, including images of final works and works-in-progress, questions and recommendations, and meta-comments. Figure 2.2 shows example images of artwork posted with the #PlotterTwitter hashtag. Most of the pieces we observed on #PlotterTwitter were abstract, generative, or math-based vector images. However, we also saw artworks where other formats were converted into something plottable, such as paintings, photographs, or 3D-scans. Examples of these pieces are shown in Figure 2.6.

In the brief 280-character span of a tweet, people who posted artwork generally shared the name of their piece along with the mediums used to create it, similar to how art is traditionally displayed in a gallery. Nearly all the tweets that posted original plots also named the machine they used—Axidraw, Makeblock, and Sillhouette are popular plotter brands we observed. For this reason, the MEX code was well-represented in our dataset, with 643 tweets. Less frequently, the tweet would also name the software tools they had used for some or all of their workflow. This was demonstrated by fewer occurrences of the codes CAD, CAP, DRI, and PLI. Machine instructions, while vital to plotting, were seldom mentioned. We suspect the artists on #PlotterTwitter generally use tools that could be coded as CAD, CAP, DRI, or PLI, but they do not frequently share them in their tweets. We also saw many cases where elaboration about tools and workflow followed the initial tweet in a thread, or was prompted by a question from a Twitter follower.



Figure 2.7 *#PlotterTwitter postcards sent by @paulrickards show that the #PlotterTwitter relationship extends beyond Twitter as physical plots are shared around the world. Postcard text: Greetings #PlotterTwitter! Here's some plotter artwork to brighten your day! This was drawn on an HP 7550+ plotter from 1986. Hope you and yours are safe and healthy. Thanks for sharing your beautiful plotter artwork. @paulrickards May 2020*

An Engaged Community Cultivates a Diverse Body of Knowledge

Our data revealed a vibrant community where #PlotterTwitter artists share plots, knowledge, and resources. They frequently recommended off-Twitter, community-created resources to each other, such as drawingbots.net or the Drawing Bots Discord server. This community relationship also extended into the physical world. During our data collection, some of the #PlotterTwitter community organized an artwork swap where they exchanged plots by mail. Other artists occasionally sent out a series of postcards with plots on them, such as @paulrickards (Figure 2.7).

We observed people on #PlotterTwitter constantly asking for recommendations for tools to use during their workflows. These questions ranged across a variety of topics, such as finding a consistent way to calibrate pen height, optimizing speed and timing options, converting vector artwork to G-code, tips and tricks for generative art, and many more. Some questions were very specific, such as one from @cicerothedoge: *"Dear #PlotterTwitter I forgot something before ordering my #axidraw... I'm in Denmark, where the alphabet depends heavily on the glyphs, æ, ø and å. None of the Hershey fonts have all 3, so I will have to create my own. Can anyone recommend a*

single stroke font creator for Win10?" Other questions were broader: *"#PlotterTwitter what software do you use for generating your beautiful paths? What are the cool programs/methods you know of? Are there any that can fill an area of an SVG with a pattern like a Gosper curve? What about ones for converting raster images? Teach me your ways! "*

These tweets reveal that #PlotterTwitter members frequently tap into the collective body of knowledge maintained by the community. This was particularly true regarding workflow minutia such as wrangling file formats. These details, which could be overlooked in scenarios like a video tutorial or instructional blog post, were often the focus of conversation. We found many instances of questions about software tasks such as generating and exporting SVGs, converting various image formats to G-code, and what machine control software was best for a certain situation.

Our data also contained many questions about machine hardware and techniques for obtaining certain physical properties: which plotter to purchase or build, how to weight pens to achieve a cleaner line, or how to create adapters for a specific pen. Further, we observed questions about materials to use with the plotter: which pens are refillable, how to properly anchor paper, how to achieve a certain effect with ink, or how to prevent a pen nib from wearing out. These questions represent a tiny selection of the topics addressed on #PlotterTwitter. Altogether, the community has experimented with many configurations and serves as a reliable pool of crowdsourced information for those who are experimenting with novel plotter workflows.

Foregrounding the Machine as a Co-creator

Instead of sharing finished artwork alone, #PlotterTwitter members frequently portrayed the means by which they created their plots. We observed many tweets which emphasized the creator's relationship with their machine and the journey that they took to arrive together at a finished product. Some people prioritized discussing the machine's actions rather than the end goal, like @265design: "*Took me a long time to set this one up but the process was therapeutic and the results pleasing.*" @qrs states, "*It's very relaxing to watch the polargraph smoothly plot the [Lorenz] Attractor on the wall over several hours*" while including an image of a hanging plotter gradually cov-

ering a large wall with curved lines. Instead of sharing the plotted work alone, it was often pictured still sitting on the bed of the plotter, or even half-finished in an "action" shot mid-plot. Artists often shared timelapse videos of the work being plotted, which were sometimes filmed from the pen's point-of-view by attaching a camera to the pen tool. These tweets highlight the importance of the machine as a collaborator in the artistic process, in stark contrast to the canonical workflow's notion of a machine as an obedient translator of digital formats to physical action.

Given the prominent role of the machine coupled with a responsive community, members were likely to share their experiences and struggles in working with the plotter itself. Specifically, defects or issues were often framed as serendipitous rather than problematic or undesirable. In one tweet, @artflychris described the defects in their plot as adding "originality and authenticity." Later, they tweet "*Tried plotting a hatch version of a geometric portrait I made. Not entirely happy with it, but what I do love are the little ink defects - I could get lost in them! I heart them defects!*" @revdancatt goes further, noting "*When your scrap paper plotting tests start to look better than your real plots. Pure alchemy going on here. Now I want to see other peoples' rejected mis-plots.*" This spawned a new hashtag within the community: *#MisPlotWeekend*.

In the canonical workflow, errors in a final product are seen as requiring a repetition of the entire pipeline to address a mistake in one part of the whole. However, these tweets suggest that community members often celebrate the holistic process. They see unintended behavior or "fails" as opportunities to generate new artworks, improve existing tools, and further explore possibilities at the digital/physical divide.

Lightweight and Flexible Design Tools are Preferred over a Monolithic Approach to CAD

While CAD was one of our more inclusive codes, with 236 instances, we found that tweets coded as CAD were very diverse in what they described. In other words, there is a large ecosystem of software used to create and manipulate digital files for drawing, and the diversity of this ecosystem defies a monolithic description. We identified three types of tools used to create and process digital designs: *direct specification tools*, *parametric tools*, and *translational tools*.

With direct specification tools, a user directly describes the geometry of an object and must manually create paths for the plotter to follow. Examples of these tools include vector graphics programs like Adobe Illustrator or Inkscape. The low-level and one-off nature of direct specification tools contrasts with parametric tools, where the latter focuses on creating a system of rules to generate paths. These rules can be adjusted and rewritten to quickly explore a wide space of designs, and they allow for exploration of emergent properties and randomness in a design system. More importantly, parametric environments can easily be shared with and remixed by other users. Some examples of these tools include conventional software like Grasshopper and Fusion360, but on #PlotterTwitter, homespun, community-created web tools dominate. On #PlotterTwitter, parametric tools often took the form of browser-based Javascript SVG generators such as @msurguy's "Flowlines" SVG generator. Encapsulated in a graphical web app, these parametric environments were easily shared through the #PlotterTwitter community.

Translational tools convert between digital formats. On #PlotterTwitter, translational tools are frequently used to transform raster images into plottable vector images. For example, @msurguy's Squigglecam generates plottable lines from a raster image, @MAKIO135 developed a photo-hatching tool, and peak-map and city-road by @anvaka translate topographic and road map information into plottable SVGs. Further examples of community-created translational tools are displayed in Figure 2.6, where artists have processed paintings, 3D-scans, and photographs into plottable formats.

Many workflows displayed in our dataset used multiple tools from these categories, adopting SVG as an intermediary format to move between different design tools. This contrasts with the idea of a one-size-fits-all CAD tool—plotter users move fluidly between parametric tools and translational tools, often only using direct specification tools at the end to "clean up" drawings, or add modifications that are specific to an individual drawing. This suggests that community members might prefer a set of smaller specialized tools over a larger tool that offers less flexibility.

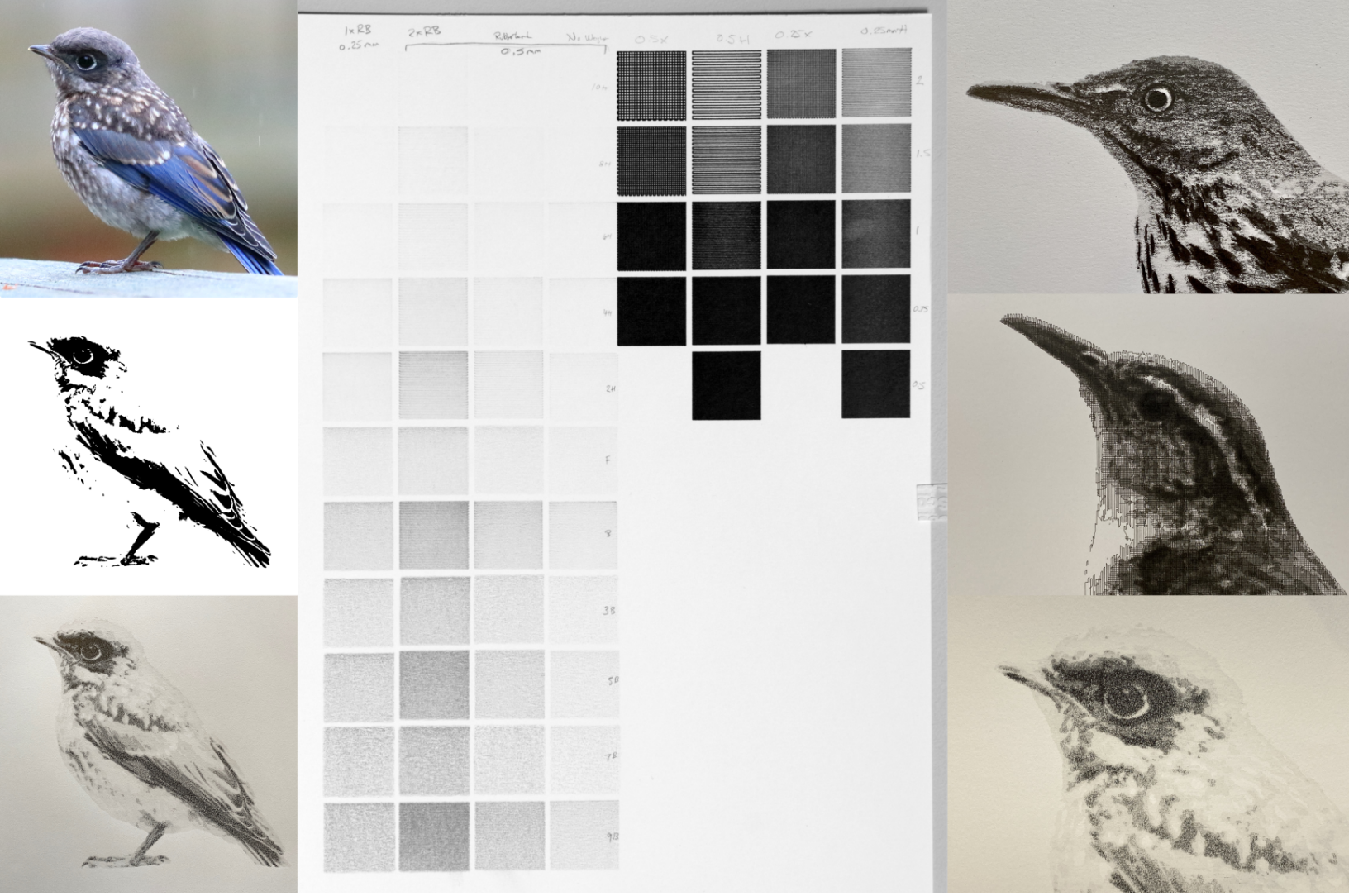


Figure 2.8 *Creating a plot often requires people to understand how their tools will perform. Left column: @FogleBird shared their process of plotting a bluebird, which involved thresholding a photograph at different values. Center column: In order to create future plots with greater dynamic range, they create a swatch palette to test different pencil hardnesses and combinations of pencil and pen. Right column: They compare birds plotted with both pen and pencil (top), only pen (center), and only pencil (bottom).*

Grappling with the Digital/Physical Divide

#PlotterTwitter community members constantly face a disconnect between digital tools and the physical properties of their machines and materials. Digital tools only show on-screen the path that the machine will eventually take, usually as a set of vector lines. However, the final result of a machine *following* this path depends on many factors, such as the acceleration and speed of the drawing instrument, the pressure, and the physical properties of the material. Vector files like SVGs still need to be modified to accommodate the eccentricities of different machines and materials. For

example, a felt-tip pen may require higher speeds than a ballpoint pen to achieve a similar line width. On-screen vector representations do not fully capture the result of a machine manipulating physical materials.

People must manually resolve any undesirable disconnects between digital formats and plotted results, as few tools exist to support this process. A common way of addressing this tension was via swatching, an iterative process of creating small samples to help the artist understand how their designs, machines, and materials will interact. They then use this information to make educated decisions about machine settings, such as the drawing implement(s) to use or the speeds and pressures to draw with.

@FogleBird demonstrates this in their process of drawing a bluebird (Figure 2.8), where they test various combinations of pen and pencil to create a dynamic range of grays in the final plot. When a community member asked about their process, @FogleBird revealed that they used the physical pencil shades in a swatch plot to make a final selection of pencils. The swatches also helped them determine how much to weigh down some pencils for ideal shading performance, which they achieved via a rubber band. If someone received the digital file that @FogleBird used to create this plot, they would still have to repeat this calibration procedure for their own machine, materials, and personal taste.

Other people on #Plottertwitter noticed and adjusted to the physical properties of their machines. @0Framerate noticed that on the Axidraw plotter, the downward pressure of the pen varies at different locations on the paper. They write, "*the #axidraw downward pressure, as you'd expect, is noticeably greater with arm extended out. Makes a difference for sensitive pastels*", showing a pastel plot with noticeably lighter sections. @revdancatt explored ways of creating rubbings or height maps of objects by placing them under the paper and using the plotter to draw over them in a controlled manner. One member remarked "*That's lovely! I really like using physical stuff to distort the digital plot.*" Another commenter described this process as *"using all the degrees of freedom"*. These examples show how creative exploration on #PlotterTwitter often spans both the digital and physical worlds.

Niche Exploration Eclipses Generic Workflows

Much of the content on #PlotterTwitter exists in a niche of its own. We were surprised by the specificity of the tasks some artists set out to accomplish. Instead of simply using a plotter to draw a vector file, we saw that artists had a vested interest in creating custom, open-ended infrastructure that supported their interests. Rather than working within the constraints of prescriptive tools that may assume a specific use-case or workflow, this approach allows for great flexibility. Critically, we noticed that this intense exploration of the functionality provided by task-specific, custom tools was often prioritized and even praised over generic approaches to plotting.

For example, @Andy_Makes developed a pipeline dedicated to converting Magic: the Gathering cards to a plottable format. In one tweet, they reflect on this process: *"I've been having a lot of fun doing plotter prints of Magic: The Gathering cards. Part of what makes it so compelling is programming my tools as I use them."* With this reflection, they also share a time lapse video of their workflow where they can be seen working via code and a visual interface side-by-side.

Later, they shared a screenshot of their tool (Figure 2.9), saying *"Since every piece of card art has a different style, I didn't think a one-size-fits-all solution made sense. So I tried to give myself a suite of image processor techniques and a simple layering system to control them."*. While the tool is specific to Magic: The Gathering cards, it allows an encapsulated set of rules to be reused, extended, and shared. @Andy_Makes demonstrates that artists are not simply creating their own tools out of necessity; instead, they take pride in the development of environments that help them explore their own, specific niches. In this way, the infrastructure artists build ultimately exists alongside their finished plotted works as a creative artifact.

Keeping with the theme of open-ended exploration, we observed community members gravitating toward existing creative coding environments such as Processing, P5.js, or Turtletoy. R was also a popular choice, particularly for people who wanted to work with statistical or geographical libraries. In particular, @MikeMahoney218 used topographical libraries in RStats to translate topographical data into a dotted, plottable height map. In this case, they worked with the `rayshader` package to isolate, clean, and process the data and create a rendered image of a small mountain range. This involved using techniques and libraries specific to the R language. @MikeMa-



Figure 2.9 *Artists on #PlotterTwitter are constantly developing tools to support their unique digital/physical workflows and vision. @Andy_Makes shared a custom tool they made in OpenFrameworks to convert a Magic: the Gathering card (top right) to a plottable format (bottom right).*

honey218 also needed to modify their workflow to isolate only parts of the mountain range above a certain height to plot. Viewed holistically, much of this workflow comprises processing different forms of data, from statistical to topographic, with plotting happening only after many stages of transforming data representations.

Other niche workflows on #PlotterTwitter incorporate additional machines. One artist, @0Framerate, followed a line of inquiry involving an embroidery machine. Before they had access to the machine, they used a plotter to visualize the eventual embroidered stitches by plotting a white gel pen on canvas. They demonstrated extensive experimentation during this time; for example, they shared that their plots needed to consist of one single line to mimic the continuous thread requirement of embroidered designs. They also documented the process of dialing in the pen's speed to achieve a realistic "bleed" effect onto the cloth that mimics the look of thread. Eventually, @0Framerate gained access to an embroidery machine and used it to actually embroider the designs they tested with the Axidraw. By simulating embroidery with the plotter, they were able to identify and address concerns before they had access to the machine.

These examples demonstrate community members' drive to develop wholly new workflows that integrate a variety of artistic mediums and existing technologies, all while continuing to promote creative, open-ended exploration. Rather than viewing these niche explorations as anomalies in a digital fabrication workflow, we might bet-

ter understand them by framing them as innovations that are expanding the total space of remixable workflows.

Discussion

Workflows Themselves Become the Medium

Altogether, we understand the diversity of #PlotterTwitter to be evidence of a pluralistic, community-driven approach to using digital fabrication tools alongside other software. Far from the standardizable canonical workflow designed for CNC in the 1950s, the breadth of work on #PlotterTwitter constantly engages with new materials, software, machines, and ways of stringing them all together. We see that many cutting-edge uses of plotters come from people venturing into unknown territory with the help of the community. This approach differs from a dominant view where careful planning is considered “best practice” in engineering domains, and suggests that we might better understand workflows—not just end products—as creative explorations that use computation as just one of a number of mediums.

In his book that explored how notions of craft carry over to digital forms, Malcolm McCullough writes that "the ability to explore a continuum of design spaces represents a fundamental advance toward using the computer as a medium [of craft]" [153, p. 172]. He adds, “tightening the loop between conception [of designs] and [machine] execution has the potential to reconcile some of the previous separation of design and fabrication that industrialization had previously imposed on craft” [p. 178].

In the case of #PlotterTwitter, a “continuum of design spaces” refers to not only different settings of a parametric design, but also to interconnected spaces afforded by different software tools, different machines, different physical materials, and the different ways the maker weaves them together. To create truly original artwork with a plotter, community members must learn how to combine disparate tools and understand their limits while still maintaining an artistic vision. Beyond the artwork shared on #PlotterTwitter, the original workflows of artists also illuminate the diverse ways humans want to work with machines.

#PlotterTwitter Defies Norms of Who Engineers and How

We view #PlotterTwitter as more than a creative exploration of digital-machine infrastructure. Instead, it is a radical alternative to understanding CNC production against its historical roots, *who* produces, and *how*. Examining the relationship in architecture between CAD and politics, Cardoso Llach writes that "software systems and numerically controlled machines are not merely "instruments," or "tools," but rather versatile metaphors reconfiguring conceptions of design, materiality, work, and what it means to be creative" [31]. Others argue that CAD and associated tools, far from being neutral representations, actively negotiate power between the designer and other stakeholders [201, 83, 57].

The canonical workflow can exclude the perspectives of those who are not familiar with CNC processes or CAD tools. This is further reinforced by proprietary design software licenses that sometimes cost thousands of dollars, placing them firmly out of reach for a casual user. On #PlotterTwitter, we see people engaging with CNC tools in a starkly different setting, against a community backdrop that looks fundamentally different than the pre-envisioned expert CAD user base.

Because no set of tools is explicitly assumed (and open source projects are actively promoted), exploration, rather than prescription, becomes the norm. The breadth of resources ensures that no single tool creator monopolizes others' creative capabilities. Community members become creators and maintainers of infrastructure and the realm of possibility expands alongside their skills and imagination. Beyond being passive consumers of fabrication tools, they are actively rewriting digital and machine infrastructure as a part of the creative process.

Implications for HCI Research on Digital Fabrication

Highlighting the plurality and radical nature of #PlotterTwitter only makes sense against the historical backdrop of digital fabrication infrastructure, which isolates the digital and physical worlds. Yet, much of the creativity on #PlotterTwitter *comes from* negotiating the physical constraints of plotters and their materials alongside their digital representations. While the SVG file format does not support most machine-level instructions, community members have already contributed tips, tricks, and hacks to

help others navigate this boundary.

Researchers and practitioners who seek to contribute to the space of tools for digital/physical creativity should respect the artistic goals that led to these hacks. This leads us to an important concession: we, as researchers and/or tool-builders, should not assume that we understand how people use (and want to use) digital fabrication technology. Further, we should not prescribe that makers use digital fabrication tools "the right way." We should instead leave room for, and perhaps even encourage, niche and case-specific interpretations.

The canonical workflow, which frames the use of CNC machinery as unidirectional and user-centered, is often useful for helping newcomers understand a basic framework of fabrication tools. However, as we have come to understand #PlotterTwitter as an assemblage of evolving practices, using the canonical workflow to theorize how people use machines obscures the many ways in which online community members are pushing the boundaries of computational craft.

While developing new abstractions to better capture the space of novel workflows might seem attractive, we argue that researchers and practitioners should avoid a top-down approach to abstraction, and instead consider how abstractions may evolve out of (and stay rooted in) current workflows. Top-down standards can prevent practitioners from working closely with machines, materials, and software tools when working across the digital/physical divide. We have already discussed such issues with the SVG file format, and this extends to the case of the STL file format for 3D printing, where, as Louis-Rosenberg writes, "file formats for additive manufacturing are lagging behind the capabilities of 3-D printing technology itself, and no one is doing anything about it" [145].

Software tools for digital/physical creativity should enable someone to access a continuum of design spaces on both sides of the digital/physical divide. The use of entrenched standards is prone to fixing one set of priorities over new ones, just as SVGs fare well for vector graphics but not for machine toolpaths. As a result, HCI researchers studying fabrication should avoid developing prescriptive pipelines for makers, instead affording makers flexibility to choose between software and data representations that suit their current needs.

Conclusion

In direct opposition to the canonical workflow, #PlotterTwitter artists create, use, and maintain open software-machine environments when creating their artworks. We see this as evidence that post-anthropocentric making is already taking place at the community level. However, research in digital fabrication within HCI largely exclusively espouses the anthropocentric view of making. Indeed, the very notion of *personal fabrication* that motivates work in the field inherently assumes that machines and their software are passive objects which faithfully obey the "person" without regard to material constraints. While this anthropocentric view is often useful, we risk limiting our understanding of the creative possibilities of digital fabrication if we do not also adopt the post-anthropocentric view.

On #PlotterTwitter, artists strive to engage with digital and physical materials at the lowest level of machine control and to share their process with others. This refutes a fundamental vision of personal fabrication that Gershenfeld formulates as being "the ability to turn data into things and things into data" [68]. Instead of becoming one and other, things and data gradually achieve new forms in tandem within a complex network of interwoven processes that individuals explore, extend, and share. We call on HCI researchers to build and engage with maker communities through a post-anthropocentric view: an understanding that digital fabrication can be more than a series of steps, that materials have agency alongside a user's goals, and that opportunities for creative exploration at the digital/physical divide are more important than seamless control.

Acknowledgements

I would like to extend a special thanks to the 29 #PlotterTwitter artists whose work appears in this chapter: @ManiacalLabs, @PhotoPuck, @artflychris, @liviopacifico, @ajpiano, @MAKIO135, @fastness, @jcupitt65, @0Framerate, @revdancatt, @265design, @v3ga, @Andy_Makes, @anvaka, @clarkkampfe, @FogleBird, @MikeMahoney218, @msurguy, @paulrickards, @qrs, @thomasp85, @cicerothedoge, @yaronski, @tasty_plots, @NadiehBremer, @kenjirototsuji, @lwrkalia, @mhkingsbury, and @plotslol.

Chapter 3

Dynamic Toolchains

Software Infrastructure for Digital Fabrication Workflows

This chapter covers Dynamic Toolchains, the first of three systems presented as part of my dissertation work. Dynamic Toolchains is an extensible, event-driven dataflow environment for building modular programs, "toolchains", which can be used to support bespoke digital fabrication workflows and explore emergent behavior at the digital/physical divide. Along with the system implementation, we contribute a series of diverse demonstration toolchains which show the versatility of this approach. The implementation of Dynamic Toolchains was heavily informed by the study of #PlotterTwitter, and two of our toolchains (watercolor painting and map plotting) are representative of work which is often shared on #PlotterTwitter.

Dynamic Toolchains is an infrastructural approach to digital/physical creativity support, complementing the next two chapters which are highly focused on the domain of knitting. A key insight gleaned from some of our example toolchains was that if given the opportunity, human intuition about relationships within a digital/physical workflow can make up for limitations of software tools, such as the inability of the SVG file format to capture the emergent behavior of watercolor paints. Supporting tight feedback loops between design intervention and outcome can aid development of intuition about digital/physical relationships. The content of this chapter was adapted from a paper published at UIST 2023 [243].

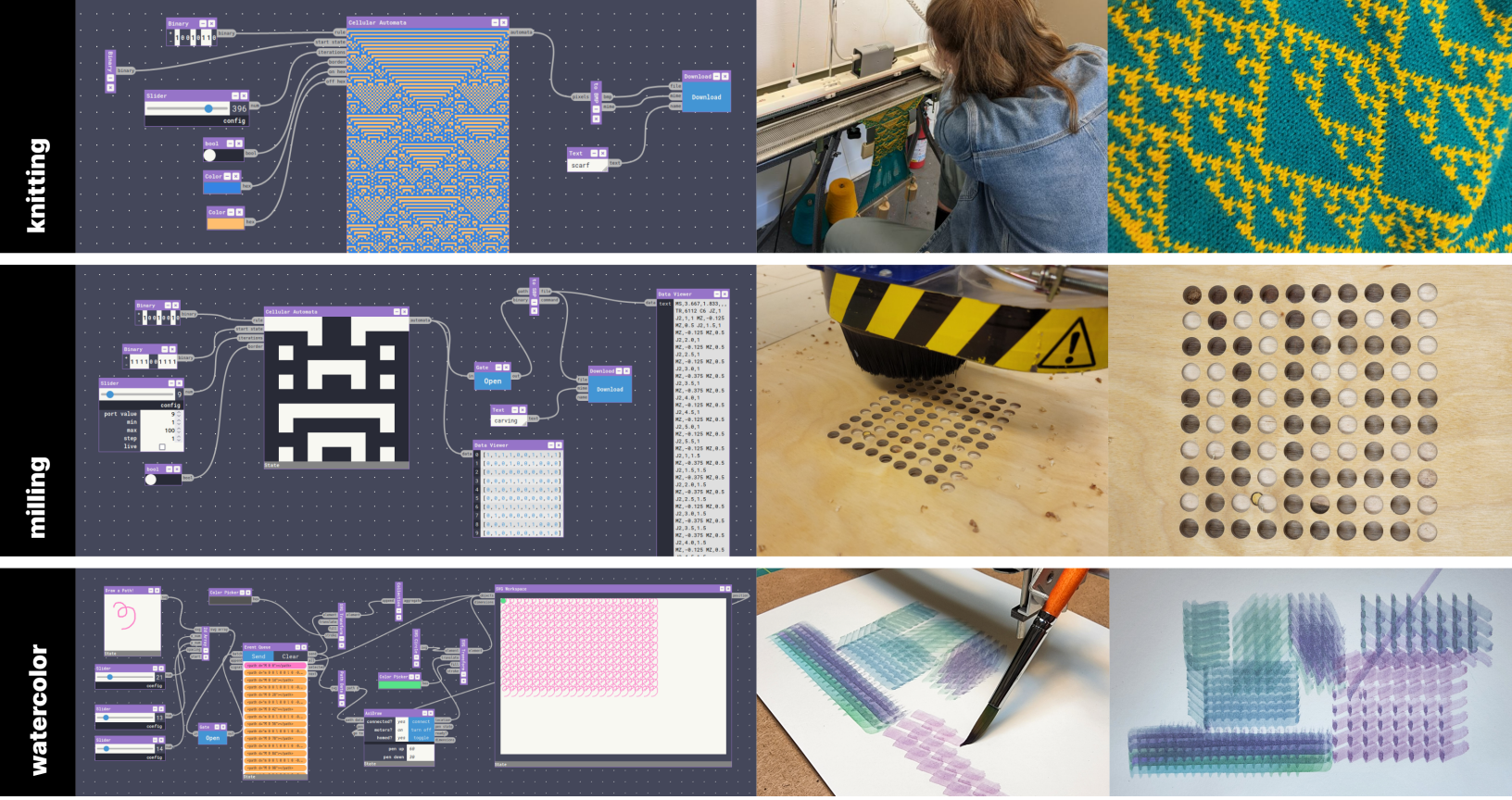


Figure 3.1 *Dynamic Toolchains are dataflow programs for digital fabrication. 1) A demonstration toolchain with modules for creating cellular automata designs and translating them to machine knitting instructions 2) Modules can easily be reused in other workflows, for example here with cellular automata computer-controlled wood milling. 3) We include modules for live machine control, supporting workflow development such as iterating on watercolor brushstroke styles.*

Introduction

Digital fabrication processes rely on computer-controlled machines that carry out instructions to create physical objects. Each object is made with a specific software and hardware *workflow*: a series of steps taken from design intent to physical product. Unlike with traditional manufacturing, these workflows enable the customization of physical outcomes just by editing code. As a result, digital fabrication workflows encompass a wide and varied range of software.

Some workflows are well-established and robustly supported by dedicated software. For example, an FDM 3D printing workflow might use the STL file format, Cura as a slicer, and an Ender 3 printer with PLA filament [17]. In contrast, new workflows that aim to capture less well-explored design spaces require process prototyping, tuning, and calibration, often requiring low-level software development and

iterative material testing [45, 35, 21, 244, 85].

HCI systems research has developed many novel digital fabrication workflows, including for exploring new materials [160, 91] and structures [166, 188, 227]. Developing novel workflows is of broad interest in and beyond HCI, whether driven by aesthetics [244, 85], material performance [19, 100, 64], or ease of use [90]. We believe that we have only begun to glimpse the landscape of possible digital fabrication workflows, and seek to support their development.

However, developing software for new digital fabrication workflows can involve complicated and tedious tasks such as wrangling file formats, deciphering proprietary machine languages, and negotiating a digital/physical divide. Even a slight divergence from established workflows involves significant time invested to tailor software for new use cases [37]. For example, adapting a robot arm to carve 3D scans for furniture production took an experienced engineer a year of "building, hacking, tweaking, and writing code" before the machine could reliably be used in practice [35]. We suspect that this difficulty in customization stems from how digital fabrication workflows rely on connecting tools and techniques from a wide range of domains, including materials science, computational geometry, and machine control. Integrating concepts across diverse domains takes time and ingenuity.

We argue that to lower the threshold to developing digital fabrication workflows, we need to facilitate system integration across domains. Towards this goal, we contribute modular infrastructure for developing digital fabrication software: ***Dynamic Toolchains***. Dynamic Toolchains (which we also refer to as "toolchains") are programs built from interactive modules such as flow controllers, visualizers, data translators, and machine communication interfaces. We adopted the term "toolchain" because we felt it was an apt description for programs where small modules ("tools") are connected ("chained") in order to support a specific use case.

We designed our system to encourage modification and extension [11, 125, 221] rather than prioritizing seamless control. Specifically, Dynamic Toolchains is designed to be portable, extensible, and compatibility-forward. New modules can be readily added to our module library and used to build toolchains in our browser interface. Modules can be built with existing Python and Javascript libraries, enabling the integration of diverse feature sets. Our system handles event-driven dataflow between

modules, streamlining communication across domains.

Instead of designing for user personas with assumed skill sets, we adopted a task-oriented approach when designing our system. We focused on supporting three different forms of user engagement [163]: 1) *Customization*, where a user explores the design space offered by an existing Dynamic Toolchain, 2) *Integration*, where a user configures their own Dynamic Toolchains from others' modules, and 3) *Extension*, where a user may develop their own modules that can be used to build new Dynamic Toolchains. Figure 3.1 shows examples of each task type: 1) customizing an existing toolchain for designing and knitting a scarf, 2) integrating the scarf design toolchain into a CNC milling toolchain for making wood tiles, and 3) extending the toolchain with modules specific to interactive watercolor and plotting. These three forms of user engagement do not clearly map to levels of expertise; for example, an existing toolchain could be complex with many interdependent input parameters while authoring a simple new module may not need more than a few lines of code.

In summary, our contributions are:

1. The Dynamic Toolchains software infrastructure which comprises (1) the toolchain dataflow environment in which toolchains are built and run, and (2) a module development framework for adding new modules to the module library.
2. A library of modules for digital fabrication including data input, flow control, visualization, manipulation, transforms, machine communication, and file export.
3. Six demonstration Dynamic Toolchains for digital fabrication workflows across different machines (knitting machines, CNC routers, embroidery machines, 3D printers, and plotters), materials (paint, yarn, thread, wood, PLA filament), and interaction types.

Our code is open-source and freely available at `https://github.com/machineagency/toolchains`. It can be installed on the three major operating systems and runs in the browser. A hosted version is available at `https://machineagency.github.io/toolchains/`. It was built with a minimal-dependency Python implementation and on top of modern web standards. We describe our system implementation in detail in Section 3 and demonstrate example toolchains in depth in Section 3. Finally, we discuss how Dynamic Toolchains fits into a broader landscape of HCI research on fabrication, workflow development, and material exploration, outlining key areas of future work.

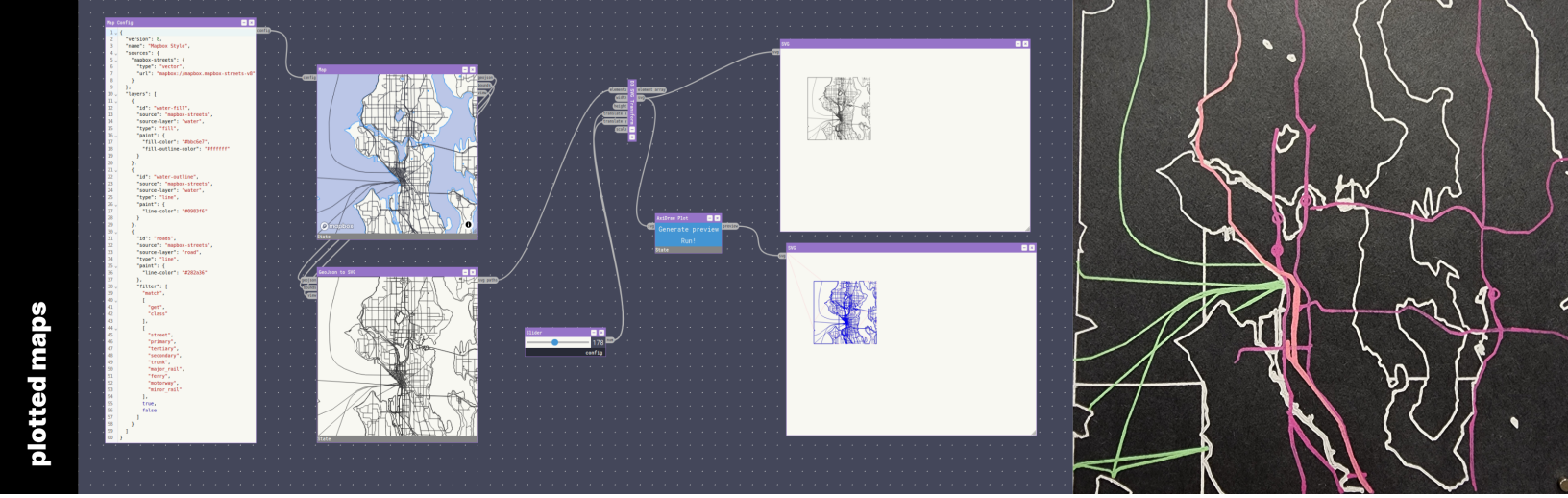


Figure 3.2 *Dynamic Toolchains makes integrating existing libraries and APIs into a modular workflow straightforward. In this map plotting toolchain (left) there are modules for interactive map selection, geographic feature specification, toolpath preview, and machine output. The plotted map (right) shows transportation routes in different colors.*

A Toolchain for Plotting Maps

In this section, we describe a motivating example of drawing geographic maps with a plotter. This workflow comprises: 1) selecting which geographic area and features to map, 2) converting that information into a format understood by the plotter, and 3) running the machine. There is no off-the-shelf solution for this; we describe developing this custom software with Dynamic Toolchains.

Plotters are popular computer-controlled machines that can draw vectors with pens and other drawing implements [244, 261, 81, 23]. Current interactive mapping interfaces such as OpenStreetMap or Google Maps store map data in plotter-unfriendly geospatial formats (e.g., SHP, GML, or GeoJSON). To plot this data, one could trace a screenshot of a browser-rendered map in a vector graphics program such as Adobe Illustrator and export the image as an SVG. However, this approach is both imprecise and limiting—vector tracing introduces an approximation error, and mapping interfaces limit customization of feature visibility (i.e. showing/hiding location names, waterways, or elevation contours) despite having access to this data. Furthermore, this workflow is highly manual—static files must be manually transferred between programs. This means that making a small change early in the workflow (such as changing the size of the map's bounding box) requires manual repetition of the entire process. Feedback is received slowly and therefore tedious to act on.

Our implementation of a map-plotting toolchain (Figure 3.2) features an interactive zoom/pan map display, customization of geographic feature visibility via a JSON

query, translation from map to vector format, plotter design positioning and scaling, toolpath preview, and machine communication. Each of these features is already well-supported by existing JavaScript and Python libraries. The core strength of Dynamic Toolchains is providing infrastructure that supports live interaction across these disparate codebases. We can (re)use code and software from external sources in an event-driven environment and rapidly iterate on designs.

For the map-plotting toolchain, we developed custom modules to interact with the following popular existing libraries: 1) Mapbox, which provides a map display and API for querying map features, 2) CodeMirror, a configurable text editor, 3) D3 [20], for translating geospatial data formats into SVG, and 4) the AxiDraw Python API for toolpath previewing and plotter control. With our module template, we only had to write a small amount of code to take advantage of features such as real-time feedback and state management. To further illustrate this, we have included our code for the AxiDraw and Mapbox modules in Appendices A and A.

Modules are connected into toolchains in our browser-based dataflow interface. Modules are live as soon as they are added to the environment, meaning that there is no compilation step required to execute a toolchain. Crucially, because our system handles communication between JavaScript and Python modules, developers do not need to implement a communication protocol specific to this workflow. As our dataflow environment supports event-driven feedback across the toolchain, downstream modules update automatically as they receive new data.

In our example map plotting toolchain, these features make it easy to quickly explore different designs—a crucial way to support creative exploration of a design space [200, 102, 92]. Here interacting with the map display and changing the feature query allows us to explore many different maps, some of which are shown in Figure 3.3.

Toolchains can easily be saved to/loaded from a JSON format and shared. This supports code reuse and portability. Our approach differs from that of other dataflow systems (e.g. Grasshopper, MaxMSP, PureData) because our execution environment supports event-driven communication between modules that are heterogeneous (JavaScript or Python) and interactive. Other approaches prioritize development of standard abstraction sets that are highly tailored to specific domains. We think that digital fabrication's diversity defies easy categorization under a unified abstraction layer, and

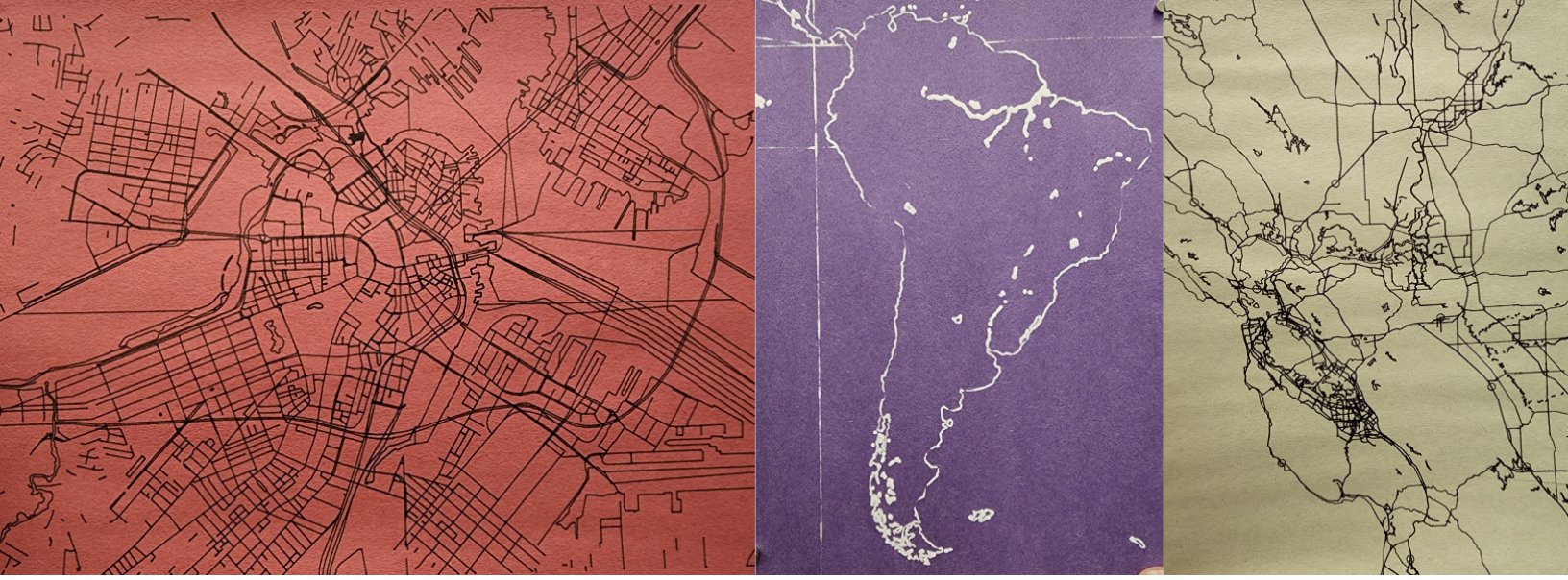


Figure 3.3 *By using the map plotting Dynamic Toolchain shown in Figure 3.2, we can quickly iterate and explore the plotted map design space. Shown here are example plots using different papers/inks with a variety of geographic features selected.*

that attempting to develop one risks constraining future developers. We argue that to support the development of digital fabrication workflows, we should instead prioritize clear paths to extension. Therefore, we contribute infrastructure for communication across heterogeneous domains. Dynamic Toolchains' key features, as demonstrated in this map-plotting example, are its combination of interoperability, extension, and real-time execution.

Related Work

We build on prior work that has contributed computational abstractions for digital fabrication. We further contribute to work that has explored how to enable humans and machines to work interactively and collaboratively. In this section, we discuss how our work bridges and extends these two streams of research.

Computational Abstractions for Digital Fabrication

Abstractions for low-level machine behavior enable developers of fabrication systems to explore niche application domains. For example, machine knitting has seen a surge of research in recent years where researchers have contributed reusable abstractions

for knitting processes, including transfer planning algorithms [139], compilers [152], and simulation techniques [264]. Building off these abstractions, researchers have been able to explore interfaces that enable humans to harness unique properties of knit objects, such as interactive textiles [147], spacer fabrics [7], and actuated objects [5]. Dynamic Toolchains provides a framework that enables abstractions, such as knitting design primitives, to be reused and combined in a live environment. Additionally, Dynamic Toolchains' built-in support for companion graphical interfaces enables abstractions to be bundled with appropriate visualizations, further aiding development of interfaces that use them by lessening development time focused on integration.

Further work has explored computational generation of machine-specific *toolpaths* in addition to domain-specific abstractions. Enhancing fabrication processes with programming concepts enables machine users to create highly customized designs that would not be possible to create in conventional CAD/CAM tools. Codeable Objects [102] accomplished this by providing a fabrication-specific extension to the Processing programming language. Extruder Turtle [188] brings LOGO-inspired turtle graphics to machine control by directly generating G-code that can be used to 3D print the path of a programmable drawing turtle. Recent work has enhanced this approach by also providing a live machine control interface. P5.fab (an extension of P5.js, the JavaScript version of Processing) enables users to control a 3D printer directly from a creative coding environment [227]. Other work has demonstrated direct machine control from a CAD environment by implementing custom components in Grasshopper, a parametric programming environment [62]. These contributions demonstrate how direct control of machine behavior, as opposed to geometry, enables a user to elicit a wide range of aesthetic outcomes from machines that would not be otherwise possible with modern CAM software. We build off of this work by bringing similar parametric complexity to a live machine control environment, where it can be used in combination with human input to quickly iterate on custom machine behavior. Unlike prior approaches, we provide a browser-based environment for designing fabrication processes, enabling us to easily extend our module ecosystem with existing JavaScript and Python libraries.

Finally, researchers have explored ways of representing and manipulating entire fabrication workflows as programs. Most similar to our work is Mods, a system for representing and sharing modular workflows [184]. Our framework expands on Mods'

approach by demonstrating how interactive feedback loops can be incorporated into a similar parametric, event-driven workflow creation interface. Another example of workflows as programs is AdaCAD. By bringing together abstractions from two very different domains (weaving and electronics) into a parametric programming environment, AdaCAD enables users to design drafts for e-textile weaving, a workflow which had no dedicated design tools [66]. By representing fabrication processes as modular toolchains, our framework supports development of interfaces where different domain specific abstractions are combined for niche applications. With this, we support future development of novel tools for other exciting niche applications.

Alternative Fabrication Interfaces

Interactive fabrication systems enable humans to control machines via live, collaborative interaction rather than a predetermined set of instructions [256]. Under this interaction paradigm, human and machines can work collaboratively to produce something that neither could in isolation. Position correcting tools [205] use human-controlled macro positioning together with machine-controlled micro positioning to produce very large artifacts with a small machine. Lucid fabrication systems mediate human and machine activity during job execution to enhance manual machine control with features such as haptic feedback [232, 231]. With FreeD [269], artists work with a handheld CNC tool to use machine precision while maintaining some aesthetic control. These collaborative approaches to machine control are an exciting area for HCI researchers to explore [116]. However, implementing these alternative approaches requires non-trivial engineering. Our system supports further research on interactive fabrication systems, as the live interfaces built with Dynamic Toolchains can take advantage of parametric module relationships to further customize how machines respond to human intervention, even while the machine is operating. Toolchains could be considered an implementation of Compton's notion of generative pipelines [39], an organized series of data transformations through which someone can explore a possibility space. Dynamic Toolchains broadens this possibility space beyond on-screen interaction into the physical world, supporting future exploration of novel applications for human-machine collaboration.

CNC machine languages like G-code are used in autonomous manufacturing en-

vironments where a full instruction set is generated before machine execution. Because interactive fabrication systems cannot rely on a predetermined instruction set, developers end up implementing system logic in higher-level languages and converting it back to machine code at runtime (e.g. as seen in implementation descriptions of [186, 165, 62, 227, 117]). By encapsulating machine communication code into modules with a generic control interface, Dynamic Toolchains can provide flexible support for event-driven machine communication without requiring developers to work directly with languages such as G-code for every new application.

Beyond one-human-one-machine interaction, HCI researchers are also exploring the forms of collaboration and interaction in digital fabrication. This includes exploring negotiations of agency [46], developing forms of craft and embodied knowledge [11], and collaborating across sites and types of machines [72]. As digital fabrication is adopted in new contexts, users of fabrication systems must contend with ill-suited machines, [125], limited resources [50], and non-human participants [143]. This requires development of new systems and approaches, and we are inspired by and seek to support these explorations with our work.

System Design Goals

In this section, we outline the design goals that guided our implementation. Our approach draws on recommendations from creativity support [200], toolkit development [75, 178], and metadesign [59]. The primary objective of our work is to support development of software for fabrication workflows. One way to accomplish this could be to develop a set of reusable abstractions which remove low-level implementation burdens from application developers and let them focus on their own creative designs [75, 252]. While powerful, prioritizing development of a unified abstraction layer also runs the risk of limiting the space of applications that can be built by end users [172, 168]. We think this is a particularly relevant concern for digital fabrication systems because they rely on integration of low-level concepts across disparate domains. Instead, Fischer et al. [59] suggests “underdesigning” systems as a way to encourage emergent behavior. Drawing from these recommendations, our main design goals are 1) supporting multiple forms of engagement and 2) features that support bridging the

digital/physical divide.

Multiple Forms of Engagement

Supporting many paths and styles of working with creative tools broadens both the accessibility to end-users and the array of designs that can be produced [200]. Mørch [163] outlines three levels of end-user tailoring: customization, integration, and extension. We aim to support these three forms of engagement. With *customization*, we aim to support using existing toolchains. Exploring a toolchain's design space should be quickly accessible through easy parameterization and iteration. Toolchains should also be easily portable across different computers. With *integration*, we aim to support adaptation and remixing of existing toolchains. Reconfiguring a toolchain should be made straightforward with a common interface for a diverse library of modules. Finally, with *extension*, we aim to support creating new modules. We support modules in multiple programming languages, require only a small amount of code to add to the module library, and limit restrictions on implementation patterns.

Features Across the Digital/Physical Divide

Digital fabrication is characterized by its crossing of the digital/physical divide: translating design intent into machine actions. Errors can be costly (in terms of time and material waste) and dangerous (in terms of human safety). Supporting human oversight of digital/physical translation has been identified as a priority for future HCI fabrication system development [90, 116, 117, 175]. Towards this goal, we will include the following three features: 1) *state management* to keep track of progressive changes to the real world, 2) *visualization* to display information for human verification and inspection, and 3) *real-time feedback* to respond to real-world events or human input during machine execution.

System Implementation

Our system implementation comprises 1) the toolchain dataflow environment and 2) a module development framework. In this section we describe the technical implementation of our dataflow environment and show how our module development framework can be used to contribute custom modules. Dynamic Toolchains is open source and available at https://github.com/machineagency/toolchains. A hosted version of the system with example toolchains and the module library is at https://machineagency.github.io/toolchains/.

Toolchain Dataflow Environment

The toolchain dataflow environment consists of 1) a Python backend which manages the toolchain data structure and available modules, 2) a browser-based interface, and 3) a server that handles WebSocket communication between them by intercepting changes to module port and state proxy objects. We adopted this full-stack approach as Python provides better support for direct access to a local operating system (which is typically required for USB-serial or integration with desktop applications) along with a robust existing ecosystem of useful fabrication libraries. JavaScript is better suited for graphical interface implementation and visualization. Our plotted map toolchain demonstrates the benefit of being able to use these two languages together: it enables us to take advantage of existing libraries in their native languages (e.g. PyAxiDraw for machine control, and MapBox (JS) for the graphical map interface).

Execution model

Toolchains follow an event-driven "push" execution model where individual modules push data to their ports (to then be received by any connected modules). The push-triggering events depend on module implementation; they could include user input, notifications from background processes, or a push event from a prior module. Modules can use threads or service workers to manage their own background processes (such as computationally intensive tasks), but downstream modules will only receive new data when it is pushed by a prior module. Other details can also be managed through module implementation. We allow for many types of modules, but being un-

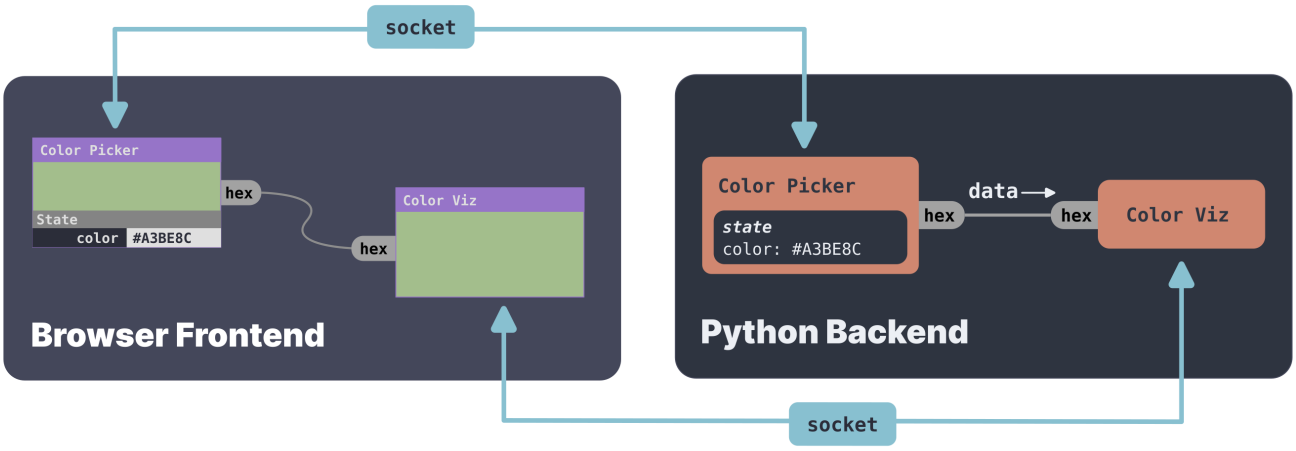


Figure 3.4 *This figure uses a trivial Dynamic Toolchain for picking and visualizing a color as an example to illustrate our system architecture. Toolchains are constructed in our browser interface by connecting module ports via pipes. Dataflow is handled in the Python backend. Our server creates a websocket connection between each module's Python class and web component. Changes to a module's port or state data are intercepted by their respective proxy objects and communicated over the socket connection.*

opinionated about permissible modules has drawbacks. For example, users could add modules that "block" the flow of a toolchain by never pushing new data.

Python Backend

The backend consists of the toolchain manager and the module library.

The toolchain manager handles creation of the toolchain data structure and manages inter-module dataflow. It provides methods for adding and removing modules, connecting and disconnecting module ports, and saving and loading toolchains from JSON. The toolchain data structure is a directed graph where pipes connect module ports, depicted in Figure 3.4. A toolchain graph can contain multiple unconnected subgraphs. It can also be cyclic, meaning that it is possible to set up feedback loops between modules. To save and load a toolchain, its current shape (graph configuration) and state (contained data) are serialized to and from a JSON representation.

The module library manages the collection of modules. It scans a directory of module definitions, which are folders containing a module's Python class, JSON configuration file, and optional JavaScript web component (described in subsection 3). The module library imports modules that have their dependencies met at runtime and provides a list of available modules to the toolchain manager.

Browser Interface

The browser interface renders a toolchain's graph structure, where module ports are connected by pipes representing dataflow connections. Each module interface is rendered inside a draggable frame component which displays the module's ports, a toolbar with "collapse" and "delete" buttons, and (optionally) an expandable panel that shows the current module state. The browser interface and custom module interfaces are defined as web components, a modern standard that encapsulates document structure (HTML), styles (CSS), and behavior (JavaScript) into one reusable element. Web components are supported across all modern browsers, meaning that our interface can be accessed across many platforms, including mobile. We use the **`Lit`** library to handle UI templating, rendering, and updating.

Flask Server, WebSockets, and Proxy Objects

We provide a minimal Python Flask server that handles WebSocket connections between the browser interface and the Python backend. The server also sets up WebSocket routes that synchronize data between a module's front- and back-ends as new modules are added to a toolchain. This is accomplished by storing module state and port data in JavaScript and Python *proxy objects*, which trigger callbacks whenever changes to the data are made in either the front- or back-end of a module. For example, a module frontend can set a new value for a state variable, which is intercepted and transmitted over the module's WebSocket route to the module's backend where it triggers the **`state_updated`** method. Changes to an outport's proxy object triggers the toolchain manager to run any connected modules' **`inports_updated`** method with the new data. This proxying behavior is handled automatically and does not require a module developer to understand or implement the data communication protocol, although it can be customized using the lifecycle methods provided by our module development framework. This feature is core to how our infrastructure enables real-time dataflow between heterogenous modules.

Module Development Framework

A module has three components:

1. a JSON configuration which defines the module name, ports, and initial state variables,
2. a Python class which customizes how the module receives and sends data, and
3. an optional JavaScript web component which is rendered by the browser interface.

As an example of how to use our module development framework for authoring new modules, we implement a color-picking module. This example demonstrates that integrating existing functionality (in this case, the browser's color-picking interface) into our real-time dataflow environment requires only a small amount of framework-specific code. Color-picking is not specific to fabrication, but the module's implementation is short enough that we can show the code here in its entirety. Implementations of more complex modules are included in Section A.

JSON Configuration

A module's JSON configuration file serves as a description of its API, which our dataflow environment uses to connect modules into a toolchain graph. The Color Picker module's JSON configuration (Listing 1) defines one outport (**`hex`**) and one state variable (**`color`**) which is initialized to a dark gray (**`#545454`**).

```
{
    "displayName": "Color Picker",
    "inports": {},
    "outports": {
        "hex": {
            "displayName": "hex"
        }
    },
    "state": {
        "color": "#545454"
    }
}
```

Listing 1 *The JSON configuration for the Color Picker module defines one outport called **hex** that other modules can accept as input.*

Class	Function	Modules Implemented
Basic Input	Input components that may be used across many workflows	Number Slider, Text, File Upload, JSON Editor, Color Picker, Binary, Boolean
Flow Control	Enable management of module communication by intercepting messages and sending them	Gate, Conditionals, Toggle Switch, Event Queue, Data Collector, Signal Button
Data Visualization	Provide visualization of various data formats	SVG Viewer, Image Viewer, JSON Viewer, Color Viewer
Data Transformation	Perform conversion between different data formats and structures	GeoJSON to SVG, SVG to PES, SVG to G-code, Extract SVG PathData, 2D Array to BMP, 2D Array to SBP, MP3 to SVG, Raster to Vector
Data Manipulation	Perform manipulations inside one data format (such as scaling or positioning)	D3 SVG Transform, 2D array, Image Resizer
Design Specification	Create geometry from various sources, such as direct specification	Path Input, Cellular Automata, Pixel Art, Interactive Map, Webcam
Machine Communication	Serial communication interface to machines	AxidrawPlot, AxidrawInteractive, Marlin-based machines
Basic Output	Save data to specified file format (e.g. G-code, BMP, SVG, SBP, PES)	File Download

Table 3.1 *Dynamic Toolchains Module Library*

Python Class

A module developer extends the base Python **`ToolModule`** class in order to customize module behavior. Following a pattern similar to component frameworks such as React, the base class provides a number of lifecycle, port, and state management methods. By overriding these methods, developers can specify how modules should respond to or trigger events. For example, the **`setup()`** method can be used to specify what should happen when a module is added to a toolchain. Our Color Picker example (Listing 2) uses the **`setup()`** lifecycle method to send the initial value of the **`color`** state variable to the **`hex`** outport when the module is instantiated. A developer might initialize a machine connection using this method (shown in the AxidrawPlot module in Appendix A). The Color Picker also updates the **`hex`** outport on changes to the **`color`** state variable. Values stored in the state are saved and reloaded with the parent toolchain.

```
class ColorPicker(ToolModule):
    def state_updated(self, key):
        self.outports["color"] = self.state["color"]

    def setup(self):
        self.outports["color"] = self.state["color"]
```

Listing 2 *The Python implementation for a ColorPicker module notifies connected tools of updates to the* ***`color`*** *state.*

JavaScript Web Component

Module developers can optionally extend our framework's JavaScript **`ToolModule`** class to specify a module's interface as a web component. The base **`ToolModule`** class integrates module-specific code with the web components standard, This means modules can integrate built-in browser APIs, events, and interface elements, such as the native HTML color input. Our Color Picker example (Listing 3) uses an HTML template literal to provide an initial value to the HTML color input's **`value`** attribute. This example also demonstrates how an event callback that updates the module's **`color`** state variable is attached to the HTML color input element's input event. Our dataflow environment then intercepts this change to the state proxy object and runs the backend **`state_updated`** method with the new color data.

```
class ColorPicker extends ToolModule {
  render() {
    return html`
      <input
        type="color"
        value=${this.state.color}
        @input=${(e) => {
          this.state.color = e.target.value;
        }}
      />`;
  }
}
```

Listing 3 *The Color Picker module's JavaScript web component renders an HTML template literal.*

As part of our contribution, we include a library of modules we developed for digital fabrication which are listed in Table 3.1. We demonstrate how these modules can be used in toolchains in Section 3.

Demonstrating Dynamic Toolchains

We developed six demonstration toolchains for fabrication tasks which are not currently supported by off-the-shelf software. Specifically, we show 1) software development with Dynamic Toolchains in different programming languages and with various existing libraries, 2) various ways of interacting with machines including interactive control and generated instructions, and 3) a variety of fabrication outcomes (e.g., embroidery, watercolor, woodwork, 3D printing). We list the modules and libraries used in the demonstration toolchains in Table 3.2. These demonstrations provide an evaluation of our system [133].

Material Explorations: Interactive Watercolor Painting

Plotters are simple machines conventionally used for drawing vector paths with a pen. By swapping the pen for a paintbrush, a plotter artist can use machine precision to manipulate materials such as watercolor paints. Current software for controlling plotters

Toolchain	Core Modules	Libraries
Knit automata (Figs 3.1, 3.9)	Automata generator, Array to BMP Converter, File Downloader, various inputs (Binary, Slider, Toggle, Text)	Pillow, Browser File IO
Milled automata (Figs 3.1, 3.8)	Automata generator, Array to SBP Converter, Data Viewer, File Downloader, various inputs (Binary, Slider, Toggle, Text)	Browser File IO
Watercolor (Figs 3.6, 3.5)	Drawing Input, 2D Array, Event queue, SVG PathData Extractor, AxiDraw Interactive Communication, SVG Viewer, SVG Transform, various inputs (Slider, Gate, Color, SVG Circle Primitive)	SVG.js, PyAxidraw, D3, WebSerial API
Plotted maps (Figs 3.2, 3.3)	Interactive Map, JSON Editor, GeoJSON to SVG, SVG Transform, AxiDraw Plot from SVG, SVG Preview, Slider	CodeMirror, MapboxGL, D3-geo, D3, PyAxidraw
Audio embroidery (Fig. 3.7)	Waveform to SVG, SVG to PES, File Upload and Download	PyEmbroidery
Velocity painting (Fig. 3.10)	G-code generator, Image Viewer, Image to 2D Pixel Array, File Upload and Download	Browser File IO, Browser Canvas API

Table 3.2 *Example dynamic toolchains we built with our infrastructure.*

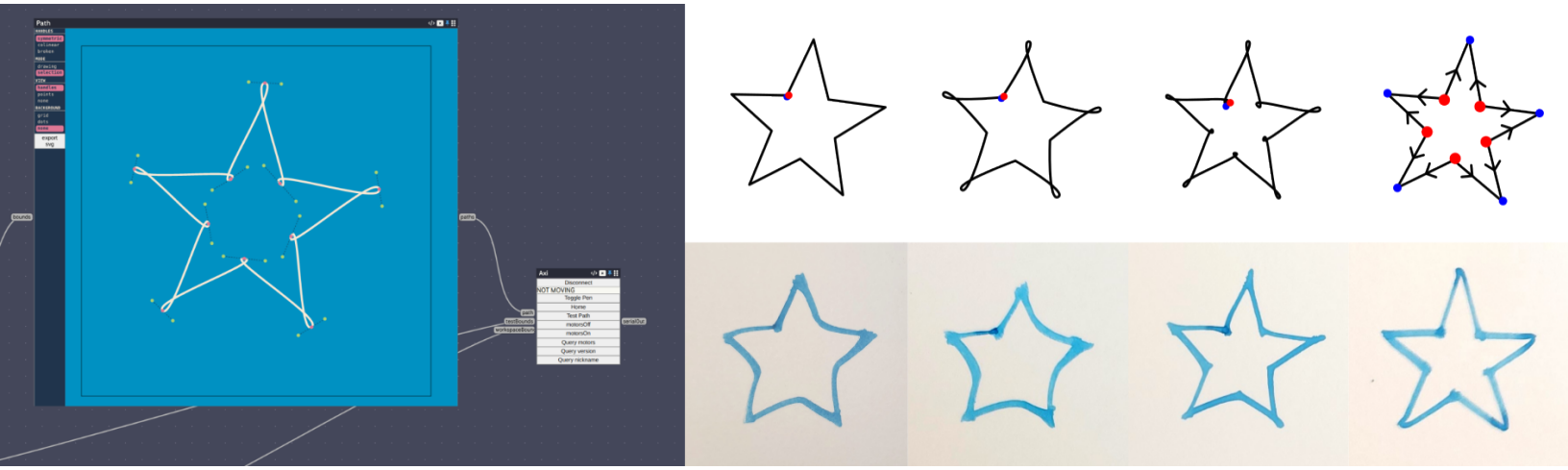


Figure 3.5 *Even simple geometry does not have a clear mapping between digital design and physical form. To build an anticipation of outcomes, we engaged in an iterative practice. Here, we use a module for manipulating vector paths to incrementally tweak the path a plotter will follow to paint a star. A simple continuous toolpath (left) results in a poor approximation of the geometry. By exaggerating the toolpath geometry (adding twists to the star's corners) or manipulating the locations of brush up (red) and down (blue) we can fabricate more satisfying outcomes.*

assumes that they will be used with a pen, and generates the machine's instruction set from a vector graphics format (typically SVG). However, painting is a fundamentally different process than drawing: the brush bends and twists, it must be periodically rinsed and pick up new paint, paint can pool and layer, etc. Further, the aesthetics of the resulting painting are highly dependent on in situ variables such as the kinds of paints, brushes, and paper used, the z-travel distance of the brush, pigment density, and more. This ultimately means that we lack knowledge for computer-controlled watercolor. We sought explore the possibilities and limitations of watercolor and plotters through iteration [125, 221].

We built multiple toolchains to help us investigate the mapping between digital vector paths and physical brushstrokes through quick feedback directly in the material (Figure 3.6, Figure 3.5). These toolchains include modules for drawing and manipulating toolpaths and a module for real-time machine connection. Here, Dynamic Toolchains functions directly as the machine controller. Machine execution can be paused, re-homed, or tweaked at any time. This way, the artist can quickly test the physical outcomes of different brushstroke toolpaths. This demonstrates how Dynamic

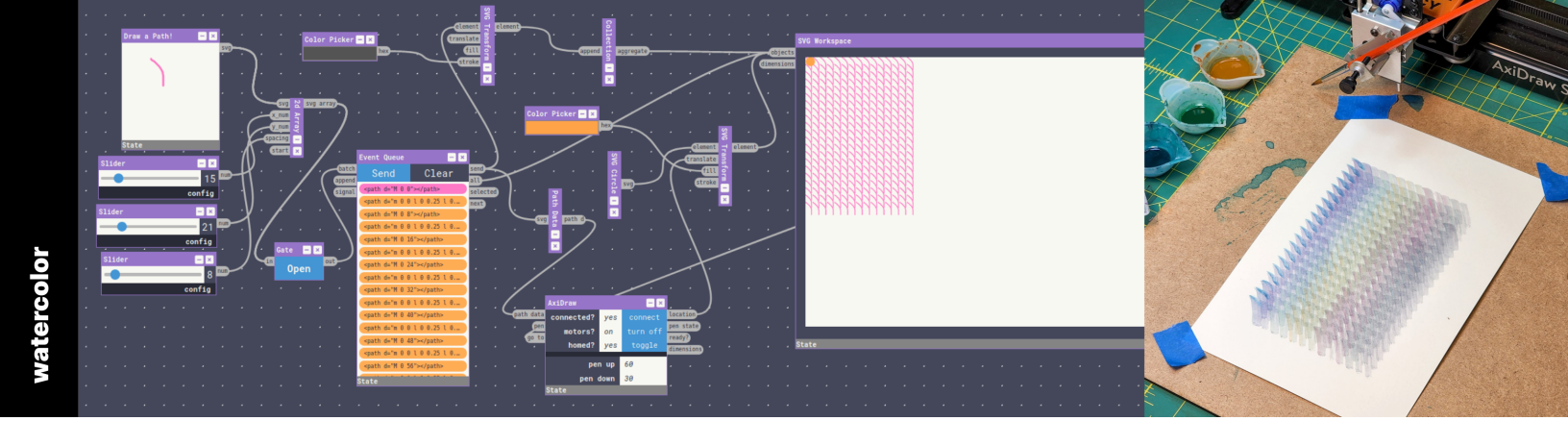


Figure 3.6 *Left: a dynamic toolchain is used to create a toolpath for watercolor painting from human drawing input, visualize the toolpath, and interactively send commands to the machine. Right: the resulting painting, where dripped colors on a brush creates a colorful gradient.*

Toolchains facilitates quick iteration in material explorations.

Connecting Diverse Inputs and Outputs: Embroidered Audio

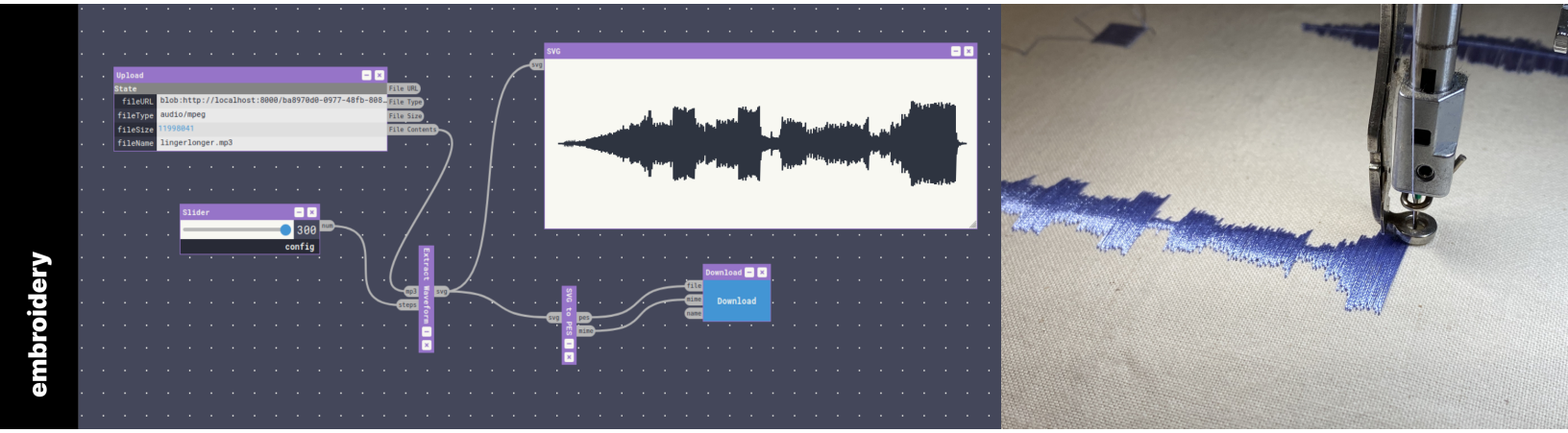


Figure 3.7 *Toolchains help us use diverse input formats to control many kinds of machines. For example, MP3 audio could be used to control an embroidery machine.*

Managing data- and file format translation is a significant challenge in digital fabrication. Yet, using unconventional, non-geometric data formats to control machines can open up new expressive dimensions [45, 173, 244]. There are a plethora of existing tools and libraries that offer a configurable way to convert between different data formats. However, using these tools typically requires intermediate file import/export steps, which slows down iteration on new designs and exploration of the design space. Here we sought to support the digital tweaking of data.

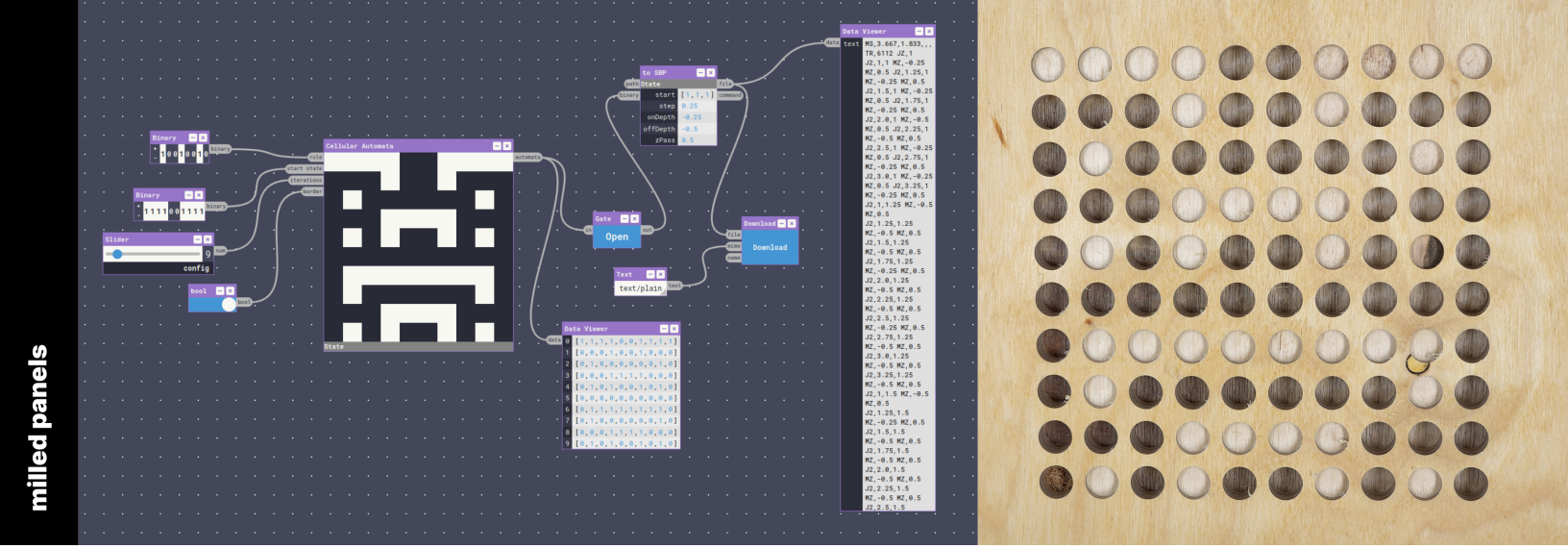


Figure 3.8 *Toolchains can be used to create lightweight parametric design systems. For example, primitive input modules (integer, string, boolean, binary) can be used as inputs to a cellular automata generator to quickly explore different patterns. Here, this functionality is used as part of a toolchain for milling a decorative wood panel by drilling holes at different depths.*

We demonstrate how Dynamic Toolchains supports unconventional input/outputs by building a toolchain for embroidering audio waveforms (Figure 3.7). This shows event-driven dataflow between modules with diverse data types. This toolchain uses two custom modules: one which uses a JavaScript MP3-to-SVG waveform library to produce an SVG from an MP3 file, and another which uses the library PyEmbroidery to convert the SVG into a format understood by an embroidery machine (in this case, PES). The toolchain also uses standard modules for uploading and downloading files, a number slider for controlling the number of waveform steps, and an SVG preview to provide intermediate visualization. When connected in the live toolchain environment, these modules work together to generate embroidery instructions from an audio file with no intermediate importing and exporting. This toolchain can easily be saved, shared, and run by others.

Multiple Levels of Abstraction: Milled Automata Panels

Many fabrication tasks are conceptually simple but would be tedious to accomplish with existing machine control software. We can have finer control over machine execution by directly building a machine's toolpath with code [227, 188]. Here, we wrote a Python script to mill a decorative wooden panel with a ShopBot CNC by drilling holes at different depths according to a 2D array of pixel values (Figure 3.8). This script only requires basic programming constructs: a nested for-loop and string ma-

nipulation. With Dynamic Toolchains, implementing this Python script as a module lets us connect it to higher-level modules which can provide varied and interactive design input. Here we use a one-dimensional cellular automata module to provide the design input. Changing the automata rule or starting state will automatically run our script, which produces machine code that can be downloaded and run on the machine. This again speeds up the process of iteration, as the script does not need to be run manually each time.

Flexibility When Adapting Software: Automata Knits

Toolchain modularity offers clear entry points for future adaptation to new use cases, which can be accomplished through development of new modules, changes to existing modules, or reconfiguration of modules within a toolchain. In this example, we adapt the automata milling toolchain (Figure 3.8) to instead produce a file appropriate for knitting with a knitting machine (in this case, a BMP file). Instead of building an entirely new system to generate design files for the knitting machine, we instead write one new module that uses Python's Pillow library to convert a 2D array of pixels into a BMP format. Connecting our new module in place of the milling toolpath generator lets us now make automata knits (Figure 3.9).

While the existing black-and-white automata functioned as a design tool, it didn't let us preview the result of knitting the patterns with different colors of yarn. Because modules are smaller in scope than all-in-one systems, this also makes it easier to modify their underlying implementation. We tweaked the automata module's implementation to add configurable color by adding two inports to the module's JSON configuration and editing two lines of its HTML template. By using a color picker input to choose colors resembling our available yarns, we were able to create a lightweight simulation of the final knitted piece. This demonstrates how Dynamic Toolchains provides flexibility when adapting software.

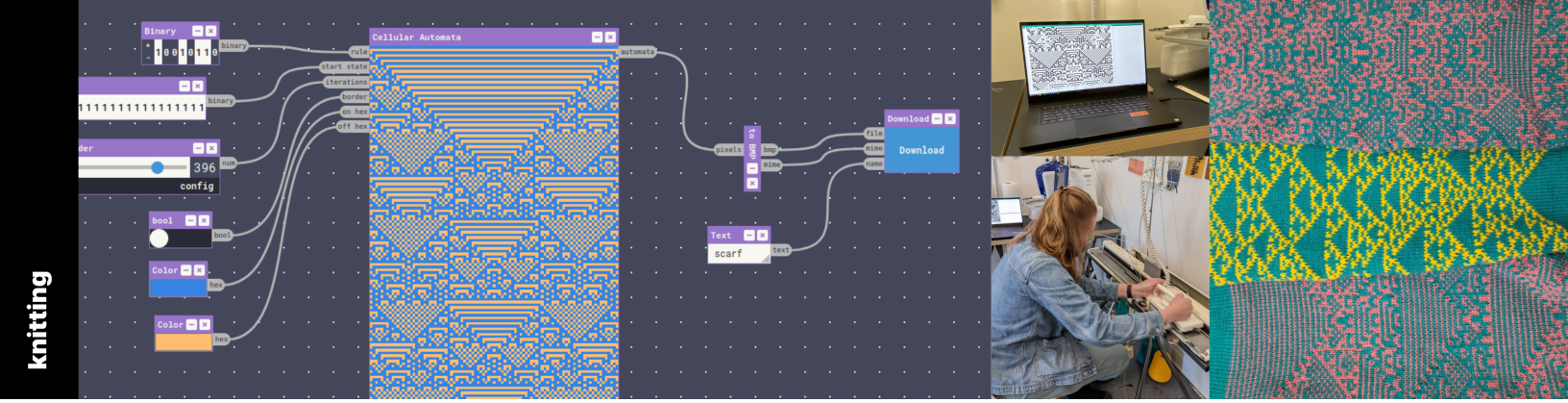


Figure 3.9 *A toolchain for designing knit scarves is built from primitive input modules (integer, string, boolean, binary), a cellular automata generator, a BMP translator, and a file download module. 2) The downloaded BMP is imported into software for a digitally-controlled knitting machine. 3) As the machine's carriage is moved, it selects which needles should knit according to the bitmap values. 4) A selection of automata knits created via this toolchain.*

3D Toolpath Generation: Velocity Printing

Dynamic toolchains are not limited to 2-dimensional machine workflows. Controlling 3D printing toolpaths at a low level allows us to create flexible and textured materials [60, 230, 166, 227]. To demonstrate 3D toolpath control, we wrote a module for 3D-printing where the printer's velocity is controlled by the grayscale value of pixels from an input 2D pixel array (Figure 3.10). This allows us to pattern 3D prints with a method known as velocity painting [255, 250]. This module uses a custom script which directly modifies G-code. By implementing this script as a module, it can be used alongside existing modules in our module library. In this case, we did not need to do additional implementation work to create a GUI for file I/O or image previewing. This toolchain demonstrates the ease of creating easy-to-use and portable software for novel workflows.

Integrating Libraries Across Languages: Plotted Maps

Our final demonstration is described in depth in our illustrative example in Section 3 (shown in Figures 3.2 and 3.3). With it, we demonstrate the ease of using external libraries and codebases with Dynamic Toolchains. Any system which is implemented across languages must implement a standard communication protocol, which can be a substantial development hurdle. Dynamic Toolchains automatically handles communication between modules written in Python or JavaScript, supporting ease of development of novel digital fabrication workflows.

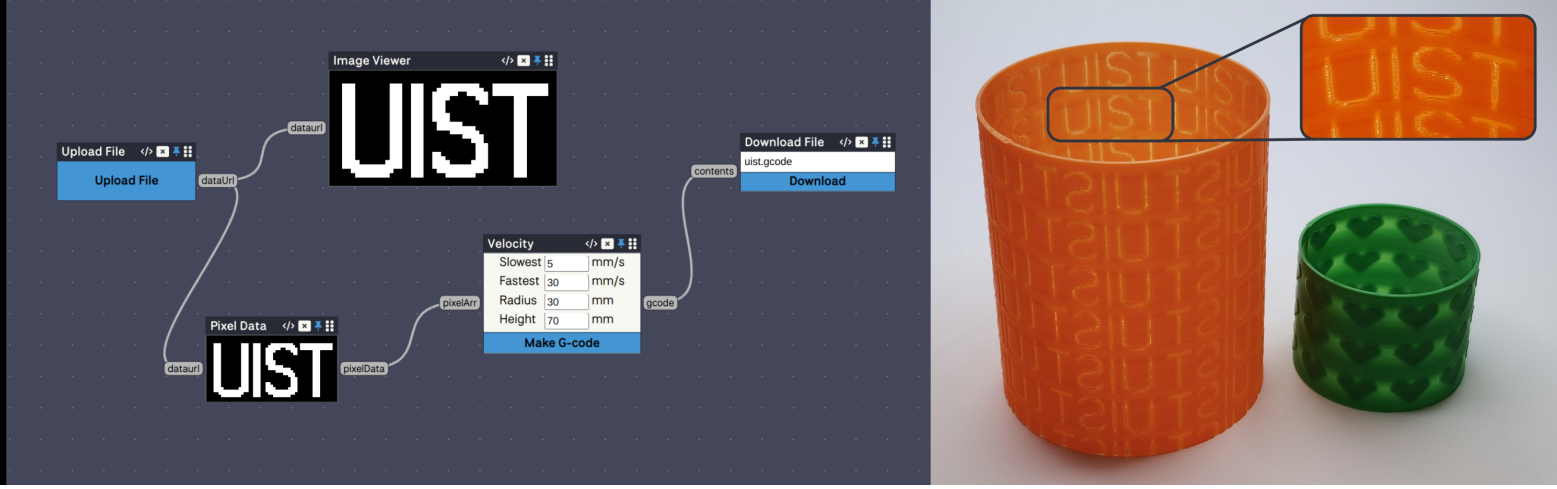


Figure 3.10 *This toolchain generates code for a 3D printer where the printer velocity, and therefore extrusion rate, is determined by the gray-scale value of pixels of an input image. This results in a patterned 3D print that is impossible to create with conventional slicer software.*

Workshop

To evaluate our infrastructure's support for extension and inform future development, we conducted a pilot workshop. We invited eight people with advanced digital fabrication skills to participate in our workshop. After introducing them to the system, we asked them to perform two tasks: First, to *install and run the development environment* on their personal computers using our documentation. Second, to *add a module to the module library*. We did not specify what the module should do, only that it should be an additional entry in the module library. We told participants that they could accomplish this by copying an existing module and modifying its content. After completing the tasks, participants were invited to stay and work on their modules as long as they wished. Depending on outside commitments, they spent 1-3 hours completing the tasks and implementing modules.

Installing and running the development environment

Seven participants successfully cloned, installed, and ran the development environment on their personal computers. These included all three major operating systems: MacOS (3/7), Windows (2/7), and Linux (Ubuntu) (2/7). The participant who was unable to install our system was trying to do so on an Ubuntu virtual machine configured for ROS development, which resulted in Python version conflicts. We could address this by including documentation for managing multiple Python versions with a package such as **pyenv**.

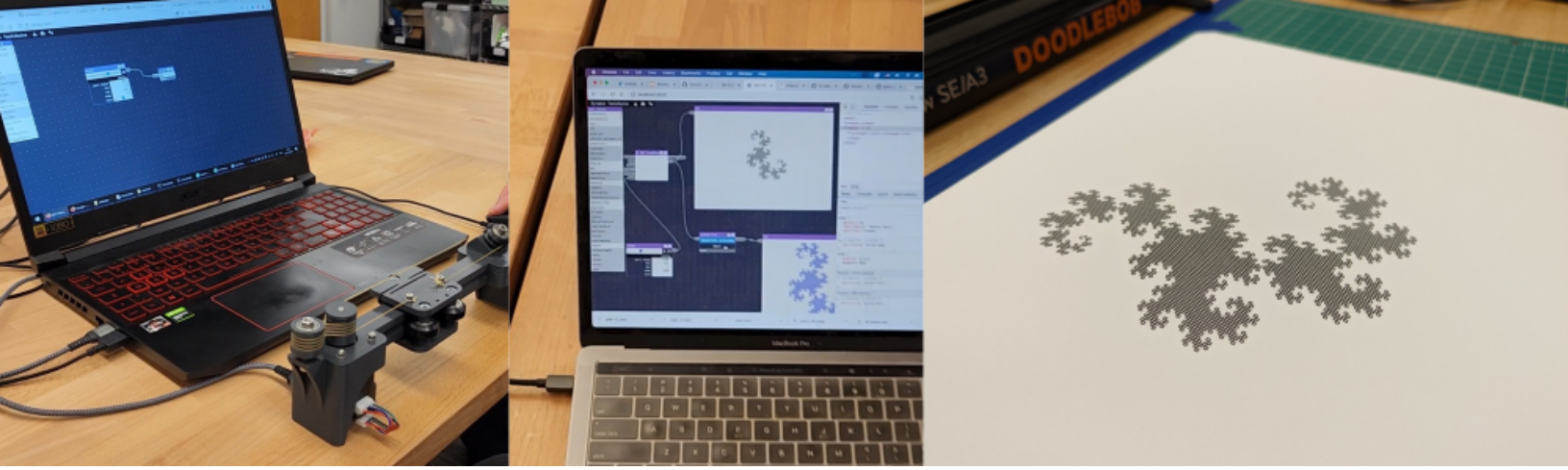


Figure 3.11 *Participants in our workshop implemented custom modules that integrated their own projects into new toolchains. From left: 1) One participant built a module for controlling their custom 3D-printed plotter. By connecting it to a numeric slider input module, the plotter moves as the slider is moved. 2) Another participant built a configurable L-system module that generates SVG fractal patterns based on an input rule. They were able to combine this module with an existing toolchain for positioning and scaling a plotted design. 3) A dragon curve that was plotted using the L-system plotting toolchain.*

Contributing to the module library

All seven participants who successfully installed and ran the system (7/7) were also able to add a new module to the module library. They did this by copying and modifying an existing module as we suggested. This required them to change the names of multiple files and classes, which some found tedious. Additionally, participants found manually editing the JSON configuration file to be tedious, and suggested that it could also be generated by a script or have a dedicated editor integrated into out browser interface.

Open-ended module development

Five participants began to implement custom behavior in their modules once they had successfully added it to the module library. Of these, three had to leave before they finished their modules, but indicated that they intended to keep working beyond the workshop and were happy to have the environment functioning on their personal machines. Two participants stayed longer to finish their modules. One integrated an L-system pattern generator that they had previously built in Javascript. The other integrated their existing Python code for controlling a custom 3D-printed plotter. Toolchains built with these modules are shown in Figure 3.11.

Overall, we found feedback from participants encouraging as it consisted of module and feature suggestions rather than challenges to our underlying implementation decisions. Notably, participants appreciated our infrastructure's support for multiple

languages, choosing to work with the language they were already familiar with or from which they could easily integrate their existing projects: *"Ports, state... I do not care what any of these things are called. All that matters to me is that I can use Python."* Another participant liked that they did not have to spend time integrating system front- and back-ends, and specifically indicated that they appreciated the existing input modules: *"It is nice to have a frontend with a slider already there; that is very useful for me. That is almost everything I need."* This indicates that our approach to full-stack module development resonated with participants, as it decreases their time spent on system integration and draw on their prior expertise when implementing modules.

During the workshop, participants suggested various improvements to development experience such as providing more example modules, designing a lightweight typing system, and implementing scripts that automate installation or generate a module template. They also suggested adding more common modules to handle tasks such as file management, image processing, and scripting. Implementing these suggested features would not require substantial changes to our system, and some were already under development. As we have personally used our module development framework extensively while creating the examples shown in this chapter, we largely agreed that all of these suggestions would be useful additions to our infrastructure. We found this participant feedback valuable and will use it to plan our future development work.

Discussion

Our example toolchains demonstrate the key contributions of our system: that the toolchain environment is flexible and responsive for handling dataflow between modules, that new module development requires minimal custom code, and finally that the library contains many useful modules for digital fabrication workflows. With our task-oriented approach to the design of our system, we sought to support 1) customization, through tweaking parameters, 2) integration, through module reuse and reconfiguration, and 3) extension, via our module development framework. Together, these three types of tasks represent the diversity of work that goes into developing digital fabrication systems, which we argue are built by integrating perspectives, tools, concepts, and practices from many domains. With Dynamic Toolchains, we sought to build an

infrastructure which supports and actively encourages this diversity.

Stepping back, an overarching goal we share with HCI fabrication research (and the broader field of HCI) is to help humans do more with machines. HCI research often approaches this goal by prioritizing usability, seamless systems, and no- or low-code interfaces [99, 51]. These values are reflected in fabrication research, where often the focus has been on developing "push-to-print" systems that enable new or novice user groups to engage with digital fabrication processes. This highlights a limitation that our approach will always have—many toolchains will be considered less user friendly than push-to-print systems if evaluated under a traditional "walk-up-and-use" metric that assumes all potential users have limited training [178]. However, novel digital fabrication workflows all start as niche explorations with perhaps only one user [244, 73]. Exploratory work is a necessary precursor to the discovery, development, replication, and refinement of new technologies. Recent HCI research indicates we have only just begun to understand what is possible with machines and materials [100, 5, 260, 122, 64], indicating that supporting future development of novel workflows will greatly benefit the field.

Our interest in supporting novel workflows extends beyond new hardware, software, and materials to new collaborations between humans and machines. Researchers have already demonstrated that systems for collaborative human-machine fabrication offer a compelling way of amplifying human creativity with machine precision, enabling humans and machines to fabricate artifacts that neither could in isolation [117, 256, 165, 232]. The terms researchers have chosen to describe this line of inquiry vary (including collaborative, interactive, lucid, compositional, and incremental fabrication), which represents the nuanced ways in which it can be approached. However, their technical implementations must tackle similar problems, such as real-time machine control, human intervention, state management, timescales, and feedback, and existing HCI theories do not necessarily extend to a digital-physical environment.

Concretely, digital-physical processes have irreversible side effects. They use and modify material, take a non-trivial amount of time, require human intervention and configuration, and come with real safety concerns. Consequently, interfaces that manage digital-physical fabrication processes cannot directly follow conventional user interface design guidelines. We were confronted with this when considering the addi-

tion of a humble “undo” button. Just as we cannot un-bake a cake, we cannot un-paint a brushstroke or un-drill a hole. “Undo” and other ubiquitous UI best practices (copy/paste, search/replace) take on an entirely new meaning in interfaces that manage the physical world.

In the future, we are excited to see how people will explore the control of digital-physical processes from a live environment using our contributions. Beyond open questions of interfaces that cross the digital/physical divide, we imagine our infrastructure will be useful for discovery of emergent material forms which are difficult or impossible to represent in a virtual environment [234, 233, 265, 235], for combining new design representations [227, 64] with new materials [160, 127, 204] and machines [63, 62, 185], and for many other explorations [222, 267, 238, 228]. We argue that iterative exploration of new domains is critical for developing the formal design primitives, tools, and techniques needed to advance fabrication research. Dynamic Toolchains’s open-source codebase, live environment, and extensible architecture makes it well-positioned to support this future work.

Ultimately, we argue that developing software that manipulates physical machines and materials is a highly nuanced, situated, and iterative process. We sought to contribute a system that not only eases traditional development burdens, but also helps people build insight and expertise to actively engage in creative exploration across the digital/physical divide.

Conclusion

This chapter presented *Dynamic Toolchains*, infrastructure for scaffolding software development for digital fabrication workflows. Dynamic Toolchains are event-driven dataflow programs built from modules such as data translators, machine interaction interfaces, and visualizers. At a high level, the infrastructure consists of 1) a dataflow environment in which toolchains are built and run and 2) a module development framework. Through six demonstration toolchains, we show how Dynamic Toolchains support digital fabrication workflows that use a wide range of fabrication techniques (plotting, CNC milling, knitting, 3D printing, and embroidery), design sources (manual path

input, geographic data, cellular automata, raster images), and interaction modalities (interactive and autonomous fabrication). Ultimately, we see Dynamic Toolchains as a way of revealing the complexity of working across the digital/physical divide such that we can explore the rich design spaces that machines and materials create.

Chapter 4

KnitScape I

Computational Design and Yarn-level Simulation of Slip and Tuck Colorwork Knitting Patterns

Dynamic Toolchains demonstrated an infrastructural approach to digital/physical creativity support, and the next two chapters now turn to a more concrete exploration of creativity support specifically in the domain of knitting. These chapters cover KnitScape, a design and simulation tool for knitting patterns. In this chapter, we expand on the idea that iterative exploration is a key method for deepening material intuition about emergent behavior at the digital/physical divide. We explore this idea specifically in the domain of slip and tuck colorwork, a deeply unintuitive color patterning technique, further emphasizing the importance of immediate feedback when exploring unintuitive relationships. The content of this chapter was adapted from a paper published at CHI 2024 [246].

Introduction

Knitting is a versatile craft which has evolved a vast design space over the past thousand years. HCI connects with textiles in many ways, e.g., through computational design tools for knits [87, 111, 106], wovens [66, 49], and ways of augmenting textiles for sensing and actuation [5, 3, 147]. Knitting is especially interesting to HCI as knit fabrics are ubiquitous, quickly fabricated, and can be algorithmically designed. Knitters today can draw on an expansive array of techniques to produce diverse textures and color patterns, intricate lacework, and three-dimensional shapes from a modest set of

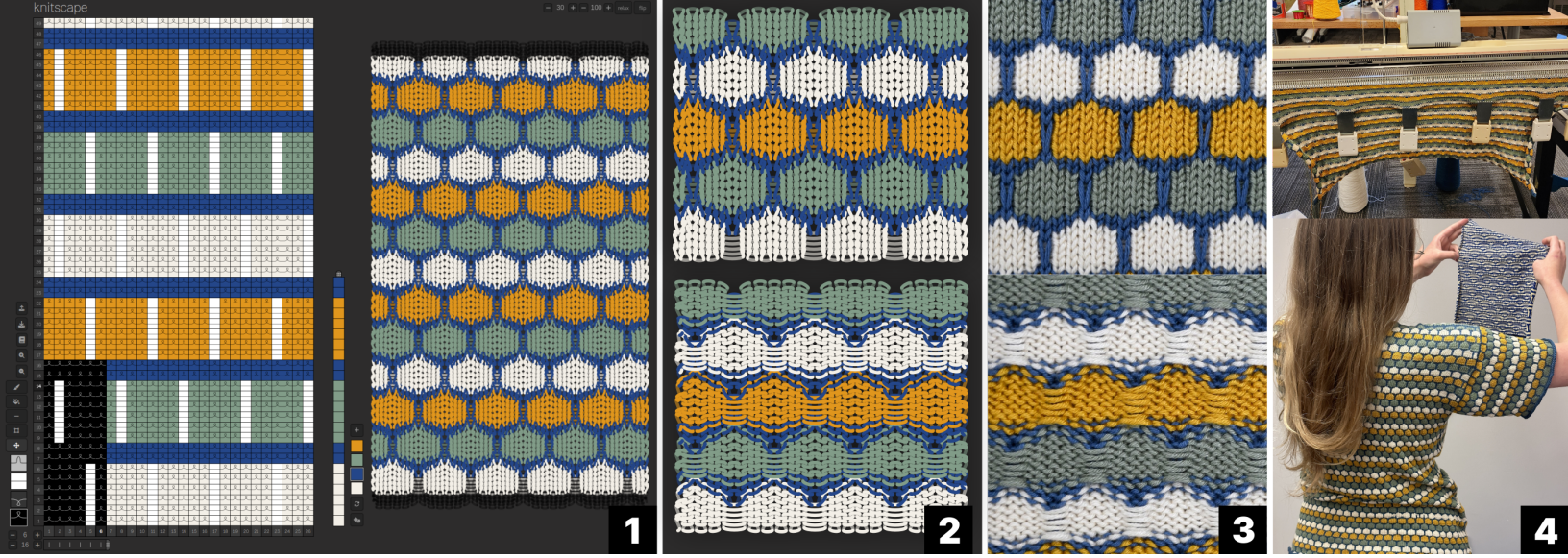


Figure 4.1 *KnitScape is a browser-based design tool for slip and tuck colorwork patterns. 1) KnitScape's interface includes a collection of pattern editing tools and a yarn simulation. 2) The back and front of the yarn simulation can be inspected during design. 3) The front and back of the corresponding knitted fabric. 4) Colorwork patterns designed in KnitScape can be adapted into garments.*

base operations. Many of these techniques rely on compound interactions between low-level operations (e.g., splitting and merging stitches [9]) to achieve an overall effect. Developing a mental model of this design space is challenging even for the most experienced knitters [254]. Even the best simulations cannot capture how a fabric looks and feels when knit.

The correspondence between knitting notations (e.g., charts) and results (e.g., knits) can be very unintuitive, traditionally requiring someone to have significant domain knowledge in order to confidently design new knit objects. This is commonly referenced as a key motivation in prior work on design tools for knitting, which has largely focused on enabling quick authoring of knit objects via high-level primitives such as 3D meshes [169, 171], sheets and tubes [151, 113], and cut-and-sew templates [115]. However, while high-level primitives can certainly be a powerful way to support goal-driven design, they also detach the digital representation from the foundational material interactions. This can prevent a designer from building ground-up intuition about the underlying processes at hand.

We see knitting's dramatic notational/physical divide as an opportunity to build tools that enable exploration of the emergent material effects of knitting. This chapter focuses on a technique for producing intricate color patterns with gentle curves known as *slip or tuck colorwork*. In this technique, described in detail in subsection 4, color patterns are produced by using certain stitches to "pull up" colors from prior

rows. Slip and tuck color patterns are particularly difficult to visualize and design because they rely on operations which deform surrounding stitches, a unique effect which cannot be captured in pixel art style tools where pixel colors map directly to stitches.

To address this, we contribute KnitScape, a design and simulation tool for slip and tuck colorwork patterns. KnitScape scaffolds exploration of the design space of slip and tuck colorwork through a split-pane interface of pattern parameters (charted stitch pattern, color sequence, and needle positions) and yarn-level simulation of the result. KnitScape's yarn simulation is intended to support an iterative material practice, helping designers choose which new patterns to invest time and resources into testing in their materials. KnitScape supports a variety of export formats and patterns can be knit by hand, on digital or punchcard manual machines, or the fully-automatic Kniterate machine.

The rest of this chapter is organized as follows: first, we provide background on slip and tuck colorwork and describe related work which we build on for KnitScape's implementation. In Section 4, we give an overview of the KnitScape interface and describe how it can be used to explore colorwork patterns, showing example outputs from a usage scenario. In Section Section 4 we describe our system implementation where we developed a simulation model which draws on a process-oriented representation of yarn topology. We demonstrate KnitScape's design space by designing and knitting a series of example swatches in Section Section 4. Finally, we discuss how our contribution fits into the HCI research landscape and what we see as key areas for future work.

Background and Related Work

In this section, we cover two areas: we provide background details on knitting to contextualize our research contributions within knitting practice, and we describe related research on knitting from fields including computer graphics and HCI.

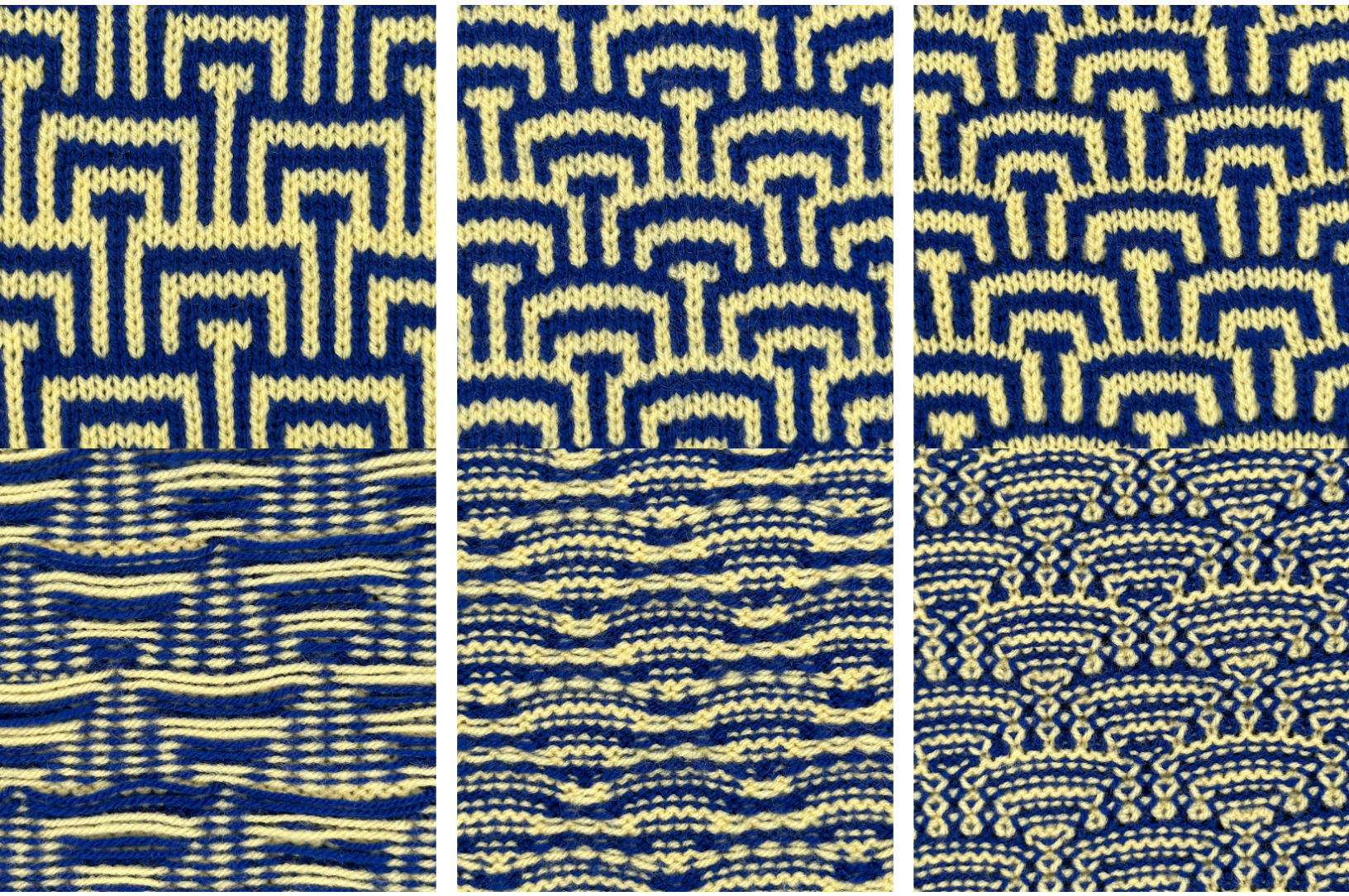


Figure 4.2 *Knitting a similar pattern with Fair Isle, slip stitches, and tuck stitches leads to very different attributes in the knit textile. This is especially apparent on the purl side of the fabric: Fair Isle (left two) has long 'floats', where the yarn is carried from one area to another, whereas for both slip (center two) and tuck (right two) stitch approaches the yarn is well integrated. On the knit side, other subtler effects are noticeable: slipped and tucked stitches can introduce curves to the pattern as they "pull up" colors from prior rows; note how the rectangular grid of Fair Isle is contrasted with the fan-like shapes of the slip and tuck pattern.*

Core concepts: Knitting with Slip and Tuck Stitches

Knit fabrics are formed via *interlooping*: loops of yarn are pulled through other loops [223]. To produce different effects in the fabric, loops can be interlooped from different sides (for knits and purls), split and combined (for increases and decreases), and crossed over each other (for cables). See the background on knitted structures in [87] for further details. There are two other knitting operations of particular interest to us: *slip* and *tuck*. A *slip* stitch (also known as a *miss* or *skip* stitch) does not form a new loop of yarn. Instead, the previous loop remains on the needle, and the yarn in work passes by leaving a *float* (a loose bit of yarn) along the fabric (Figure 4.3, center row). The more stitches which are slipped consecutively in a row, the longer the resulting float will be; the upper limit varies from project to project but is typically around four in a row. A *tuck* stitch also leaves the prior loop on the needle, but adds another loop to the needle without pulling it through.

We find it helpful to think of slip and tuck stitches as "pulling up" yarn from lower rows by holding a loop on a needle, with yarn in interstitial rows either being "slipped" or "tucked" behind the held loop. When the interstitial rows are knit in a different yarn color than the loop that is held, color patterns are produced as the held loop obscures the slipped or tucked yarns (see the vertical blue lines on the front fabrics in Figure 4.3). Because slipped or tucked stitches are held for multiple rows, they pull the rows they connect together, creating a local deformation in the fabric. A clear example of this is shown in the hexagon pattern of Figure 4.1: a loop is stretched via a vertical column of six slips. This "pulls" the rows of dark blue yarns together, transforming what looks like a rectangle in the charted view into a hexagon in the resulting knit. Increasing the number of rows of knitting that a loop is held for exaggerates the deformation created. The longer the loops are held, the stronger and stretchier a yarn must be to prevent breaking. To see how this effect differs from a much more common colorwork technique, stranded or "fair isle" colorwork, see Figure 4.2. On our manual knitting machine, we could successfully tuck up to six times on the same needle, but this limit will again depend on the project, yarn, and method of knitting.

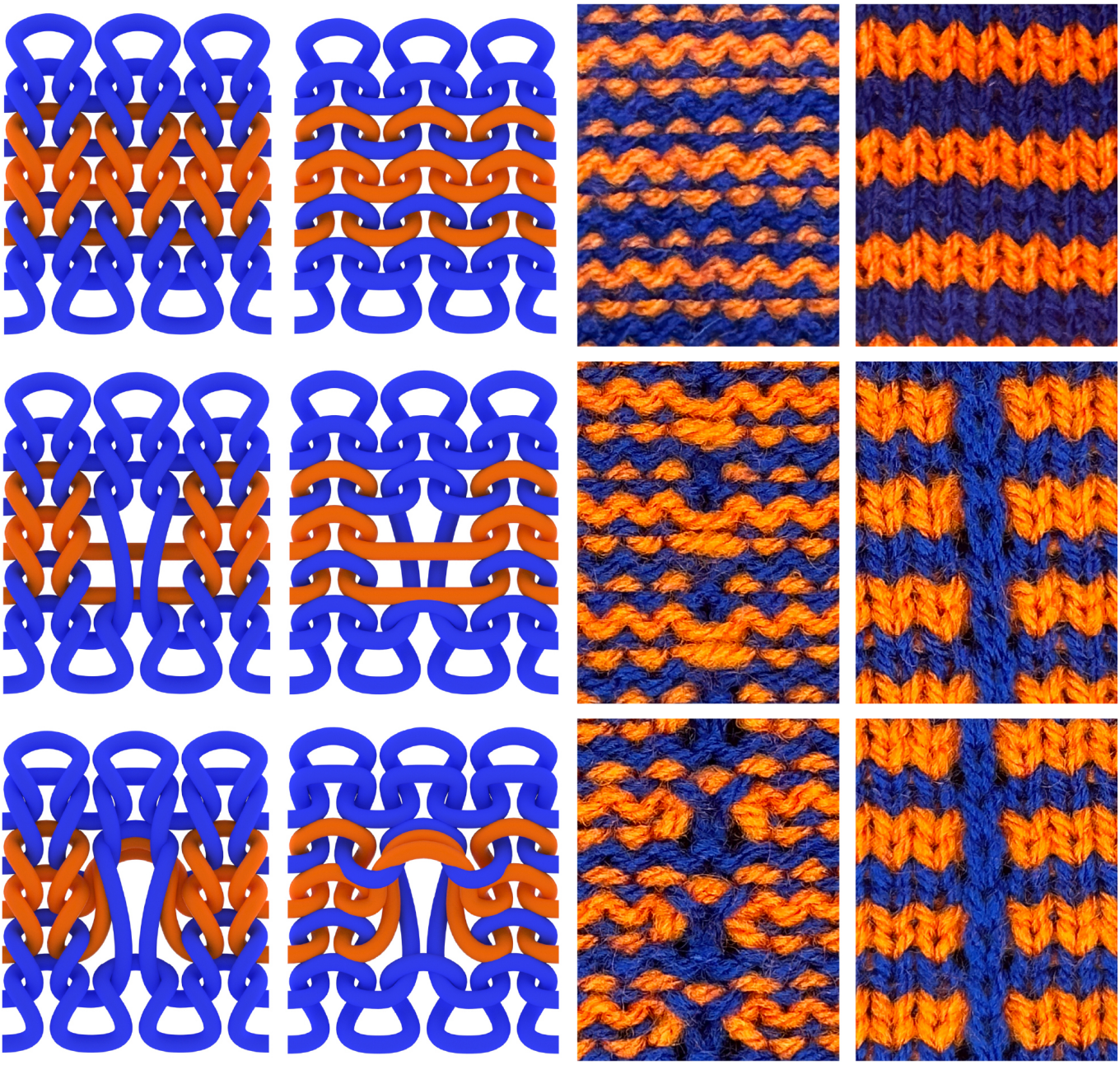


Figure 4.3 *Top: Knits, Middle: Slips, Bottom: Tucks. From l-r, each row shows 1) graphical front, 2) graphical back, 3) photographed back, ("wrong side") 4) photographed front ("right side"). For the slipped and tucked stitches, blue from the lower rows is "pulled up" by slipping/tucking two consecutive rows of orange, creating vertical lines of blue on the front.*

Existing Design Tools for Textiles

Researchers across domains have demonstrated a broad interest in supporting textile fabrication processes through development of new design tools, graphical models, programming languages, and fabrication techniques. This chapter focuses on knitting, but past work has looked to support other methods of textile design and fabrication such as sewing [132], weaving [49], quilting [130, 131], plush toys [164], and embroidery [55, 81, 134]. Each of these domains offers further opportunities to enhance textiles with sensing and other electronic components to create e-textiles [70, 71, 183, 154, 3, 248]. There is a clear need for new hardware and software tools to support designers looking to work across the traditionally disparate domains of textile and electronics design [66, 30, 28]. While we do not focus specifically on e-textiles here, we think that building lightweight and accessible design tools for knitting is a prerequisite step toward development of future tools that open up design and fabrication of knitted interfaces to broader audiences.

Knitting Research in HCI and Computer Graphics

An area of focus in HCI and computer graphics research has been on developing tools for creating fully 3D knit objects. The challenges center on translating 3D objects to knitting instructions, where 3-dimensionality is achieved by increasing and decreasing the number of stitches at different locations. For example, work by Igarashi et al. [94, 93] generates hand-knitting patterns for stuffed animals and provides previews using texture maps of stitches and a simple physical simulation of the global shape.

With the advancement of whole garment knitting machines much effort has focused on digital design workflows to generate machine instructions without requiring the user to have extensive domain expertise. McCann et al. [152]'s knitting compiler automatically outputs low-level machine instructions from tube and sheet primitives, Narayanan et al. [169] develop a fully automatic approach that works on arbitrary triangle meshes. Kaspar et al. [113] offer an interactive web interface for customizing garments composed of parametric knit templates. Stitch meshes [264, 257, 258, 259] were introduced as a modeling framework for representing the layout of stitches on 3D surfaces. In contrast to the focus on 3D shaping in prior work, our design tool highlights design decisions in colorwork.

Researchers have also explored methods of supporting design, simulation, and fabrication of knitted textures. Hofmann et al. [87] introduce KnitGraphs, a data structure for representing the loop structure of knitted textures that can be used to design and manipulate patterns for both hand and machine knitting [87, 88]. As part of their design and yarn-level simulation tool for periodic patterns, Leaf et al. [129] use a tile-based model where charted knitting instructions are encoded in rectangular tiles. Kaspar et al. [114] automate pattern design from images of a real knitted sample using a deep learning framework. Albaugh et al. [8] investigate how to support hands-on machine knitting by augmenting a manual knitting machine with sensing capabilities to provide real-time visualization of the machine's state, including a mass-spring simulation of the fabric as it is being formed.

Substantial fundamental contributions have also been made in effectively programming knitting machines. Lin et al. [140] propose a new mathematical basis for machine knit objects along with formal semantics that is among the first to cover the complete space of v-bed knitting machine programs. Yu and McCann [263] developed a system for real-time coupling between knitting code and a topological visualization. Narayanan* et al. [171] introduced a visual programming interface. The focus is on creating 3D shaped objects, but also supports colorwork achieved by intarsia, Fair Isle, and plating. Related work has also dealt with transfer planning to assess efficiency of machine knitting instructions [139].

Hobbyist and Professional Tools for Knitting

Tools available to today's hobbyist and professional knitters range from free or cheap chart-based stitch editors, to paid software for garment design and pattern drafting, to enterprise design software for factory knitwear production.

Free and low-cost tools for knitting predominantly take the form of chart-based editors where symbols representing base knitting operations are placed on a grid. While the default symbol sets have some consistency across tools, there is no standard set of symbols and most tools allow knitters to customize the available symbols. There are many chart-based editing tools available across desktop, web, and mobile platforms. Examples include Stitch Fiddle [58], Stitchmastery [226], and Chart Minder [158]. Notably, Stitch Maps [26] breaks grid-based presentation of stitch patterns by drawing them as they may appear along the fabric (i.e. by following curving courses

and wales). Many of these knitting-specific design tools have dedicated groups on the fiber arts-oriented social forum Ravelry [198]. As chart-based knitting notation is easily transferable to spreadsheet or pixel-based paint software, knitters are also able to adopt programs such as Numbers and Gimp for advanced pattern design use cases [40].

The next tier of paid software for knitting combines garment-level patterning and stitch-level planning, and is aimed at dedicated hobbyists or small businesses. These tools include Cochenille Design Studio [95] and Designaknit [146], both of which have a one-time cost of a few hundred dollars depending on selected features. These programs offer a suite of editing interfaces that support various aspects of the knitting process, including garment-level sizing and styling, stitch-level editing and tiling, and pattern-level drafting. Designaknit, which seems to be most popular among today's community of manual machine knitters, also offers an interactive knitting interface for both hand and manual machine knitting to help knitters keep track of pattern progression and notify them of color changes, increases, and decreases. However, these tools do not incorporate knit simulation or visualization beyond grid-based image mapping of knit/purl stitch faces. This means that while the colorwork patterns we demonstrate with KnitScape could be designed in their stitch-level symbol editors, there would be no visualization of the resulting knit.

Finally, Shima Seiki and Stoll, the two main manufacturers of industrial knitting machines, both provide a comprehensive suite of software for industry knitwear design and fabrication. These packages include a robust set of stitch- and knit-pattern design tools and templates, advanced fabric simulation and rendering, stitch pattern libraries and yarn databases, and manufacturing and supply-chain oriented features (e.g., for yarn sourcing and factory floor management). While these tools are undeniably powerful, they are prohibitively expensive and specifically aimed at design for fully-automatic machines, making them inaccessible to the vast majority of hand and manual machine knitters.

Simulating and Rendering Fabrics

Modeling the appearance and behavior of knitted fabric is an area of intense interest [33], and has advanced in parallel with new techniques in computational geometry and dynamic simulation. Early work saw the introduction of geometric principles and representations for knitted structures [128, 167]. Eberhardt and Weber [52] extended work by Breen et al. [22] to simulate dynamics in cloth draping for knitted fabrics. Rémion et al. [211] modeled stitches as splines and animated control points, demonstrated on a regular grid of stockinette. Meißner and Eberhardt [156] developed *KnitSim*, which first introduced a process-oriented system to build a topological model, and was a precursor to *TopoKnit* [109] which is the basis for the topology model used in KnitScape's simulation.

Kaldor et al. [107, 108] published foundational work in yarn level simulation that computes the contact response of yarn-yarn collisions. They sought to reproduce material behavior of yarn dynamics for applications in animation, whereas our focus is on interactive design. The stitch mesh representation was introduced by Yuksel et al. [264], which simulated local deformations through a combination of mesh-based and yarn-level relaxation. Continued work extends to 3D shaping [257], knittability [258], wearability constraints [259], and an augmented representation with inter-face dependencies [171]. While the stitch mesh representation has produced significant advancements to pattern design and simulation, some limitations exist in producing the topological model, notably tuck stitches are not represented. Cirio et al. [36] define a new representation for knitted textiles based on persistent but slidable yarn-yarn contacts to support efficient simulation. Their approach produces realistic renderings for texture patterns, as well as simulation of macroscopic behavior.

Closest to our work in terms of user functionality, Leaf et al. [129] do yarn-level simulation of swatches coupled with an interactive design interface. Their simulation focuses on texture arising from the stitch pattern compared to our emphasis on colorwork. They incorporate GPU implementation for their physical simulation whereas we use an abstracted representation that is lightweight and can be run in a browser. Similar to stitch meshes [264] they represent stitch patterns as a grid of 'knit-tiles' to encode topology. While this works well for knit, purl, and slip operations, it is difficult to extend to tuck operations which create complex topological changes to surrounding

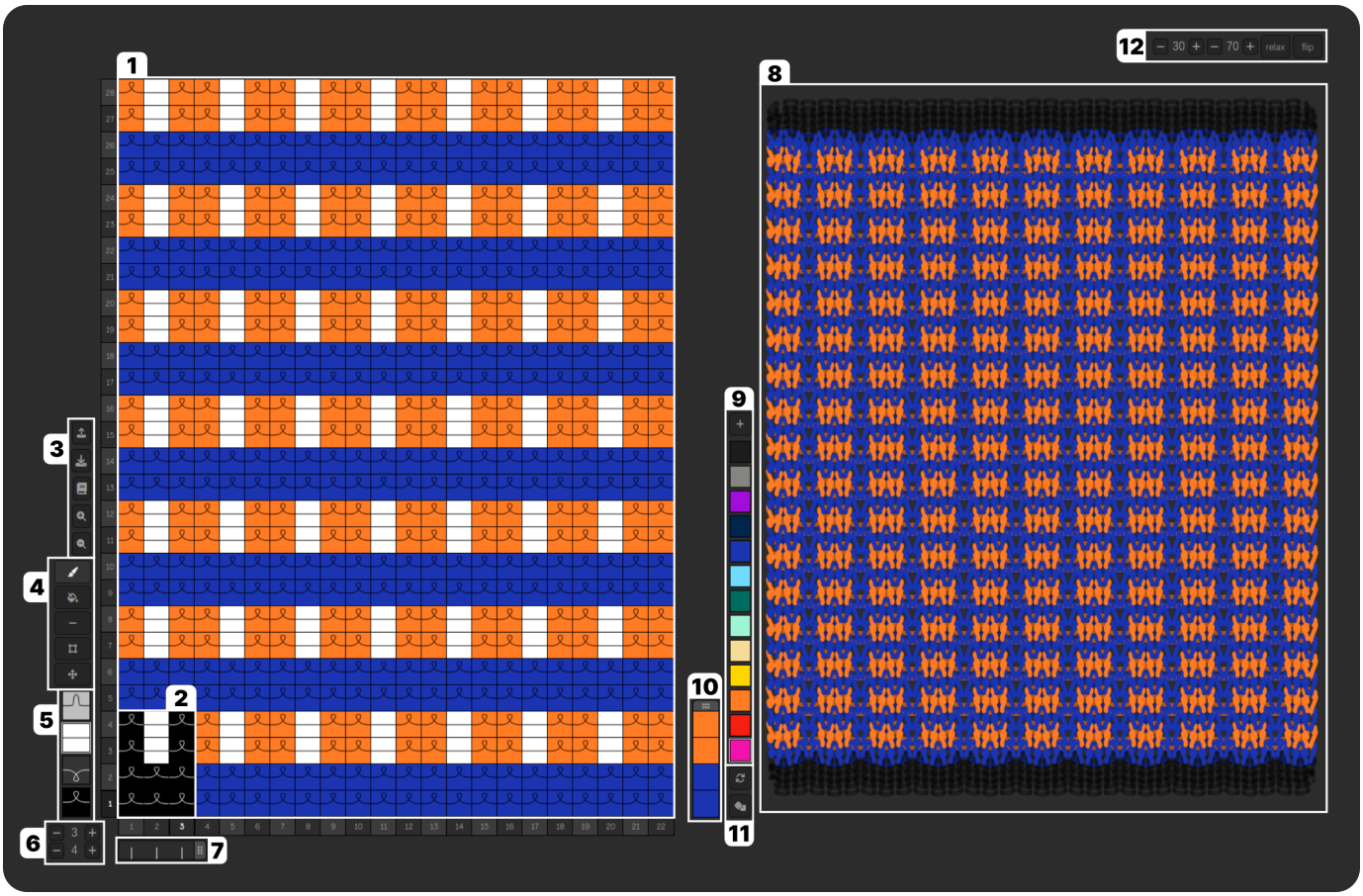


Figure 4.4 *The KnitScape interface. 1: Colored operation chart 2: Base operation repeat 3: File menu 4: Paint tools 5: Stitch Palette: Tuck, Slip, Purl, Knit 6: Repeat size 7: Needle position editor (in and out of work) 8: Swatch simulation 9: Color/Yarn library 10: Color sequence editor 11: I'm feeling lucky (randomizes colors) 12: Simulation size.*

tiles.

Swatching with KnitScape

In this section we describe how someone may use KnitScape as part of a knitting workflow to explore the design space of slip and tuck colorwork patterns.

Russell is a knitter who recently bought some new yarn in a pleasing palette: cream, magenta, and navy. Russell is not yet sure how they would like to use this yarn, but thinks that it would make a nice colorwork project. They decide to become

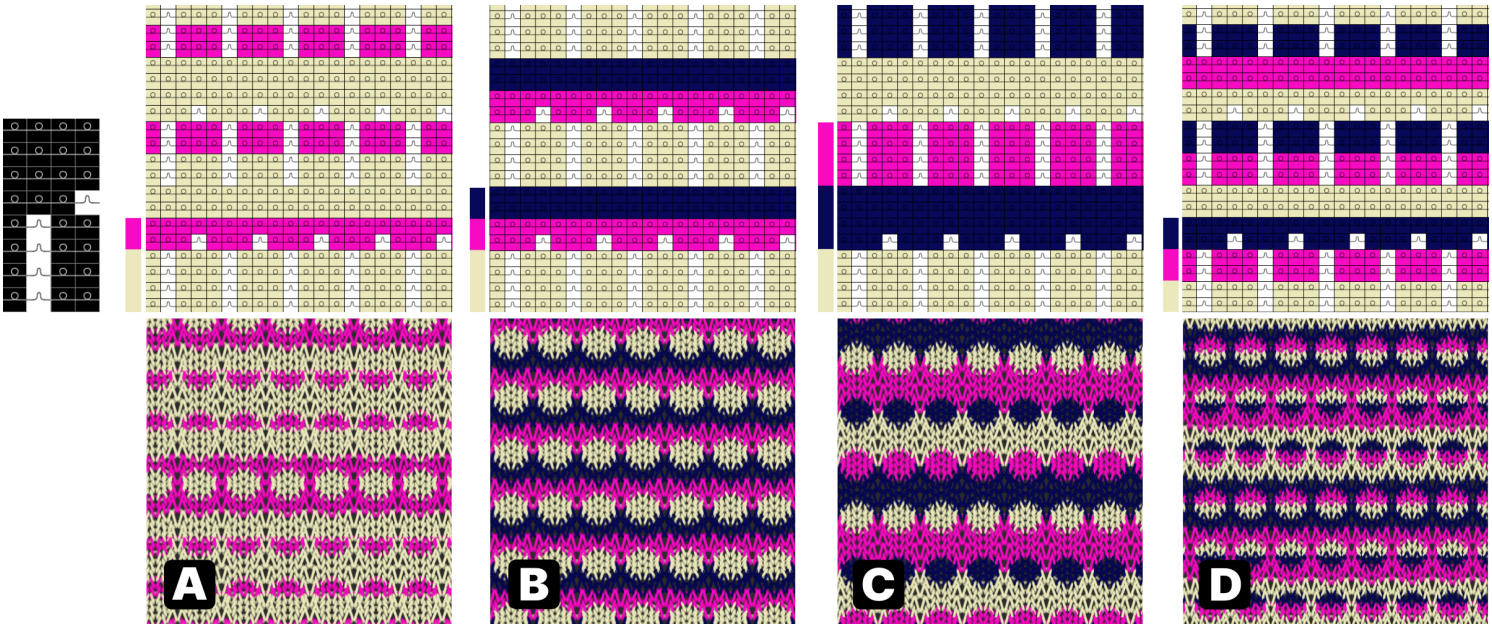


Figure 4.5 *Each of these swatches use the same stitch pattern and different color sequences. Even for a small 4x8 stitch pattern (far left), different color sequences can produce dramatically different patterns. KnitScape supports rapid exploration of this relationship by regenerating the yarn topology in response to color sequence or stitch pattern changes.*

familiar with the yarn by knitting some test swatches. Swatching is central to the knitting process, but knitting a swatch takes time and Russell has a limited amount of new yarn to experiment with. Russell decides to use KnitScape to explore different color patterns and select which ones to test in the new yarn. They open the KnitScape website, which randomly selects a simple blue and orange pattern from a folder of example patterns. The KnitScape interface is depicted in Figure 4.4.

Russell begins by adding their new yarn to the library of yarn colors (Fig. 4.4, 9). They paint their cream and magenta yarns onto the color sequence editor (Fig. 4.4, 10), which is immediately reflected in the yarn simulation view (Fig. 4.4, 8). They drag the corner of the black-and-white stitch repeat (Fig. 4.4, 2) to create a four by eight repeating pattern. They use the stitch editing tools (Fig. 4.4, 4 and 5) to draw a simple starting pattern (Figure 4.5, far left). This starting stitch pattern and color sequence is tiled across the chart view and shown in the yarn simulation (Fig. 4.5, A). They run the simulation by hitting "relax". They watch the tucked loops contract, pulling on surrounding stitches. The rectilinear chart will, when knit, form a series of ovals.

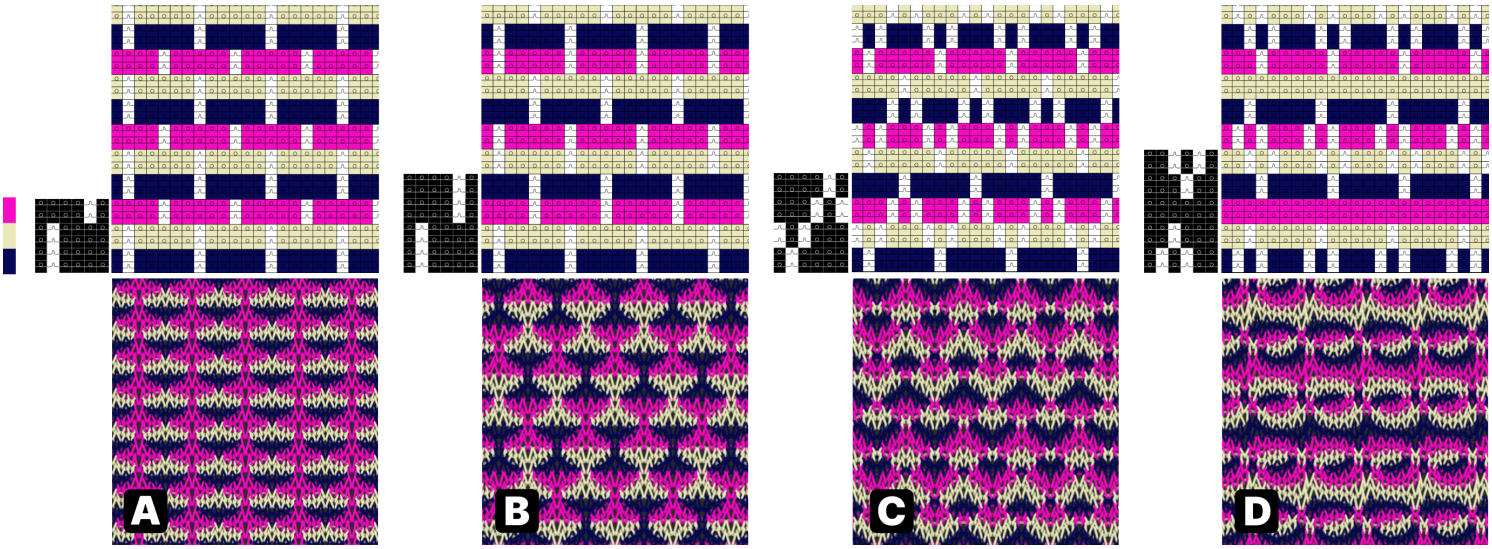


Figure 4.6 *Each of these swatches use the same color sequence and different stitch patterns. As in Figure 4.5, even for a simple six-row color repeat (far left), different stitch patterns (black and white) produce dramatically different fabrics. The interplay of stitch pattern and color repeat is key to slip/tuck colorwork. Figures 4.13, 4.14, and 4.16 show examples of more complex color sequences and stitch patterns.*

Russell begins to explore changes to the color sequence. First, they drag the color sequence editor up to match the height of the stitch pattern repeat and add their navy yarn at the top (Fig. 4.5, B), producing a regular pattern of cream-colored ovals. They further extend the color sequence and make the blocks of color larger, now changing every four rows (Fig. 4.5, C). Now that the stitch pattern and color sequence heights do not match, the rows of ovals cycle through navy, cream, and pink. Russell notices this and decides to see what would happen if the color sequence was instead shorter and colors changed every two rows (Fig. 4.5, D). The oval effect is still present, but the ovals are now bisected by color. Russell sees a slight resemblance between this pattern and a creature-capturing device from one of their favorite video games, and downloads the KnitScape workspace JSON to save it for future reference (and makes a note to buy red yarn).

Russell decides to leave the color sequence alone for a bit and moves to editing the stitch pattern repeat. First, they resize the repeat, making it shorter and wider (Fig. 4.6, A). Next, they make it taller and finish the vertical column of tucks on the right (Fig. 4.6, B). Russell draws a new stitch pattern by doodling a zig-zag (Fig. 4.6, C), but they find the resulting knit pattern to be a bit busy for their taste. They increase the repeat size again and modify it slightly in order to space out the tuck stitches (Fig.

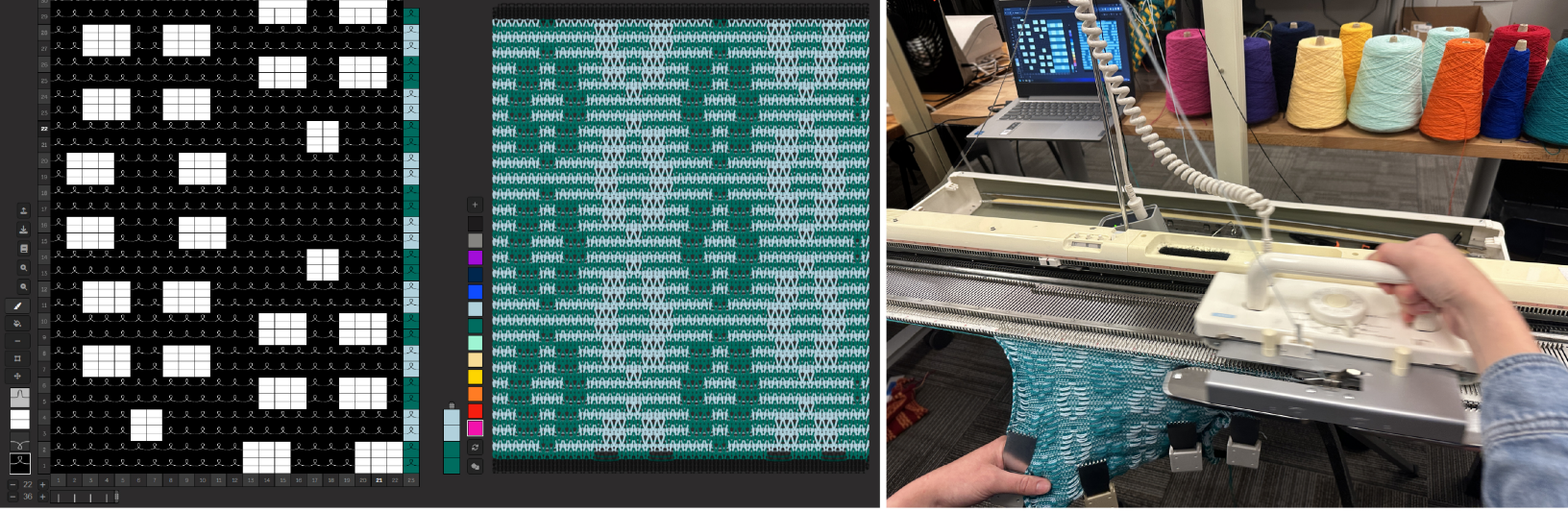


Figure 4.7 *We used a Silver Reed SK840, a manual knitting machine with digital stitch patterning, to knit the example swatches shown in Section 4. We added colors corresponding to our available yarns to the KnitScape yarn palette.*

4.6, D). They are surprised by the twisted spiral effect of this pattern, and decide to use it as their first swatch. Russell saves this workspace and exports it to the TXT format supported by their Silver Reed SK840 manual knitting machine.

In the following sections, we provide more detail on KnitScape's implementation, features, and example outputs.

System Implementation

KnitScape is an open source client-side webapp written in JavaScript. In this section we describe KnitScape's input parameters, yarn-level swatch simulation, and output data formats. Links to the live site, source code repository, and demonstration video can be found on the project page: `https://depts.washington.edu/machines/projects/knitscape/`.

KnitScape Parameters

In KnitScape, colorwork patterns are built from three elements: the base stitch pattern repeat, the color sequence, and the needle positions. Each of these is specified via a resizable bitmap editor and stored in a JSON format (Listing 4).

```
{
  "stitchRepeat": {
    "stitches": [0, 2, 0, 0, 2, 0, 0, 0, 0, 0, 0, 0],
    "width": 3
  },
  "yarnPalette": ["#ff7b24", "#0f4bff"],
  "colorSequence": [1, 1, 0, 0],
  "needlePositions": [0, 0, 0]
}
```

Listing 4 *The JSON data format for the KnitScape swatch shown in Figure 4.4 comprises a yarn palette (orange and blue), the stitch repeat, the color sequence, and needle positions. The numbers in the stitch repeat refer to indices of the stitches in the KnitScape stitch palette. Here, 0 is knit and 2 is slip. Similarly, the color sequence consists of indices of yarns in the yarn palette.*

Stitch Pattern Repeat

The base stitch pattern repeat is a rectangular grid of knitting operations. In the KnitScape interface, the base repeat is shown at the bottom left of the chart view and tiled to fill the chart. We chose to overlay the base repeat on top of the colored chart because base repeats can have dramatically different sizes and aspect ratios. Importantly, stitch symbols –not colors– are painted onto the operation chart interface; colors are assigned separately via the color sequence editor (described below). Stitches can be painted anywhere on the tiled chart, and the change will be mapped to the appropriate location in the base repeat.

KnitScape currently supports four operations: Knit, Purl, Slip, and Tuck. Further stitches (e.g. increases, decreases, and transfers) could be added in the future to support shaping and lace, but are not necessary for KnitScape's target application of slip and tuck colorwork patterns. In the tiled preview, knit and purl stitches (black and dark gray) are colored according to current color sequence. Slip and tuck stitches are left uncolored to represent that the yarn currently being worked will not be shown in that row on the right side of the fabric.

Color Sequence

The color sequence represents the color of the yarn assigned to each row of the operation chart. Yarns are selected in the yarn color library and painted onto the sequence editor. Editing an existing color in the yarn library also updates the color sequence,

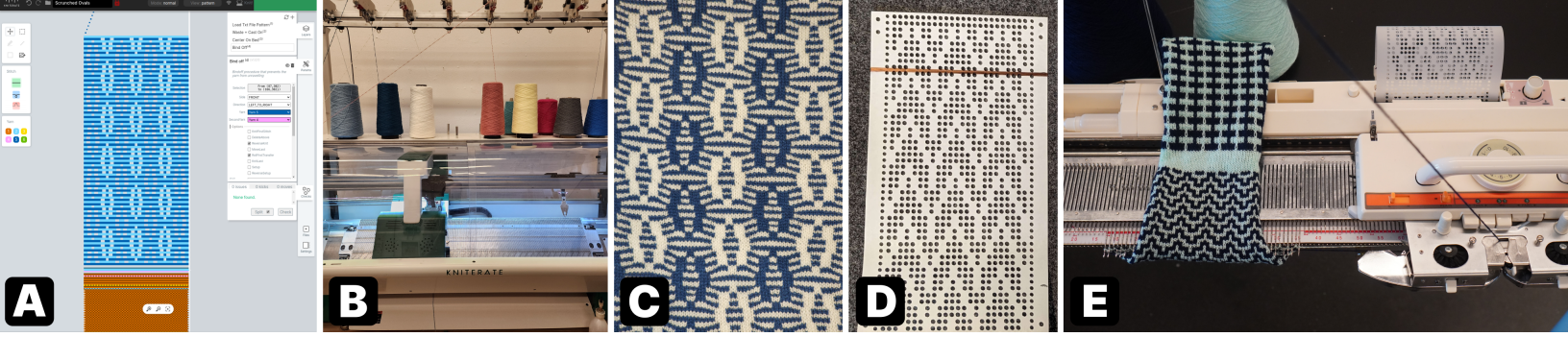


Figure 4.8 *KnitScape exports formats supported by very different machines. KnitScape exports a TXT format supported by the Kniterate design app. Once the swatch pattern is imported, cast-on and bind-off instructions can be added using one of Kniterate's built-in features (A). The swatch can then be knit on the Kniterate machine (B). Kniterate uses a finer yarn than we used to knit our demo swatches in Section 4, but the deformation is still apparent in the resulting swatch (C). KnitScape also exports an SVG format which can be used to cut punchcards out of Dura-lar film with a craft cutter (D), and used to control a punchcard machine such as the Taitexma TH-860 (E).*

operation chart, and yarn simulation. The color sequence is repeated up the stitch preview chart and does not need to be the same size as the base operation repeat. It is resized by dragging the handle at the top of the sequence editor up or down to the desired sequence length.

Needle Positions

On manual knitting machines, needles can be removed from work to create ladders in the knitted swatch (discussed further in subsubsection 4). KnitScape also includes a needle position editor to define a needle position sequence. Needles are toggled between working position (I) and non-working position (O). Like the color sequence editor, the needle position editor is resized by dragging the handle left or right to the desired sequence length. The sequence is repeated across the swatch and represented in the chart view by a column of greyed-out squares.

KnitScape Output

KnitScape is flexible when it comes to knitting technique, and supports a variety of export formats. We have designed and knit patterns on a manual machine with digital stitch patterning (Figure 4.7), an automatic flatbed machine, and a punchcard knitting machine (Figure 4.8). KnitScape currently supports the following output formats:

1. the raw workspace JSON (described in Listing 4) which KnitScape also supports as an upload,
2. a black-and-white BMP of the stitch repeat (a format accepted by electronic pattern controllers on manual knitting machines),

3. a TXT format specific to SilverKnit, the electronic punchcard system used by our knitting machine (a Silver Reed SK-840),
4. a TXT format supported by the Kniterate design app,
5. an SVG format which can be used to cut 24-stitch punchcards for use with punchcard knitting machines (tested on a Taitexma TH-860),
6. a PNG of the colored chart view, for use in hand knitting, and
7. an SVG of the current yarn simulation view (which we used to create the figures in this chapter),

Besides exporting for machines, the resulting charts can also be hand-knit. Knitscape therefore supports a broad range of common knitting machines and techniques. Because KnitScape uses a simple rectangular chart representation, adding additional output formats for other machines (e.g., Shima Seiki) is straightforward.

KnitScape Swatch Simulation

Data structure for yarn topology

KnitScape uses a yarn topology representation based on *TopoKnit* [109]. Accurate yarn-level topology is necessary for our target design space as it permits tracing yarns to keep track of color properties. Furthermore, the combined geometric layout of yarn segments and topology of yarn crossings supports our efforts in simulating mechanical properties of the knitted output. Figure 4.9 shows an example topological layout for a pattern consisting of knit and tuck stitches.

The data structure is a directed graph, where nodes represent yarn crossings (termed *contact neighborhoods*), and directed edges represent the yarn segments. Yarn crossings are significant as they specify locations where a yarn is 'fixed' by interlock with a yarn loop and is a fundamental requirement in simulating mechanical behavior. Nodes have valence 2n, where n is the number of yarns crossing that point. In addition to the topographical layout, nodes are classified as *potential*, *actualized*, or *unanchored*. For example, a series of knit stitches will have nodes of valence 4 that are actualized, representing the interlocking contact between the two yarns where the loops cross over each other. Conversely, a tuck stitch may have valence 6 or higher to

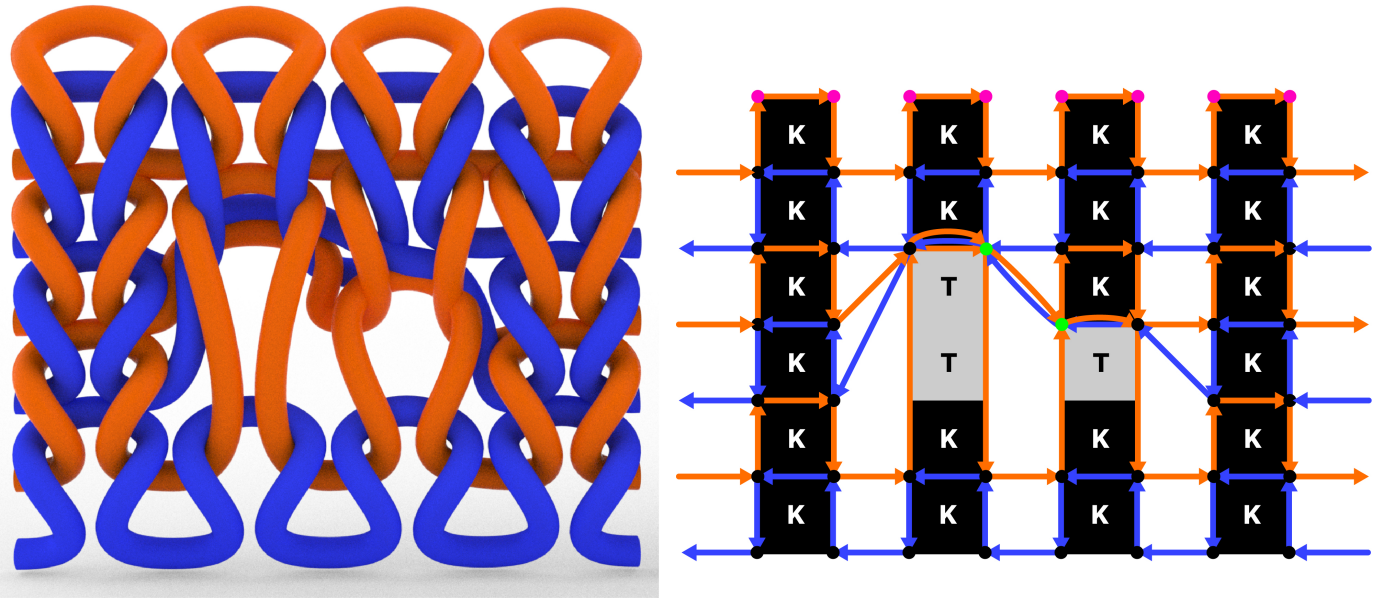


Figure 4.9 *(Left) A sample pattern of knit and tuck stitches, and (right) the corresponding topology graph. Directed edges represent yarn segments and encode orientation: orange edges are rows where the yarn was carried left to right, and blue edges right to left. Black/grey faces indicate which operation was performed (in this case knit (K) or tuck (T)). Nodes correspond to yarn crossings. Black nodes are actualized, pink are potential, and green are unanchored.*

account for the yarn(s) pulled up one or more rows (see Fig. 4.9).

Populating the data structure and generating the topological structure is process-oriented and involves parsing the stitch instructions row by row in the same order they would be executed. Stitch types and node classifications are established as each row of the pattern is evaluated, and *movement vectors* keep track of how these properties may shift as subsequent rows are knitted (e.g., tucks being pulled up one or more rows). For complete details we refer to the TopoKnit paper [109].

Yarn-level swatch simulation

We contribute a yarn-level simulation which builds on our implementation of the yarn topology data structure. We had three goals when developing KnitScape's yarn simulation. First, it should be browser based and written in client-side JavaScript. Second, it should have minimal dependencies to support future reuse and extension, by us and hopefully others. Third, it should be fast enough to provide rapid feedback when used alongside our stitch pattern editing interface. Before settling on our eventual SVG yarn-level simulation, we implemented two iterations (canvas and SVG) of a tile-

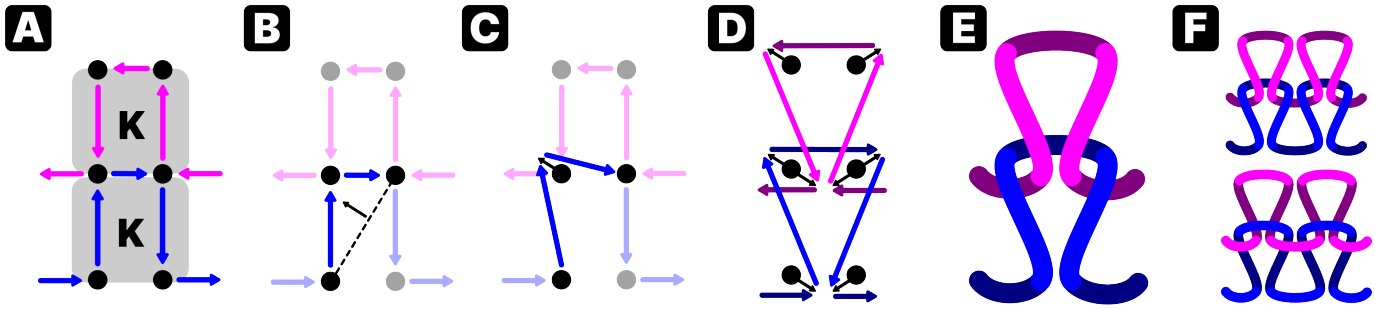


Figure 4.10 *A) We start by laying out nodes corresponding to the contact neighborhoods on a grid. B) To determine the offset for a yarn passing through a node, we determine the normal to the line connecting the nodes on either side. C) The updated position for the yarn. D) The offsets are calculated for each yarn segment endpoint. E) Splines for each yarn segment are drawn through the offset nodes. F) The yarn segments are separated onto two layers depending on whether the contacts were anchored with a knit (legs on front layer) or a purl (legs on back layer).*

based swatch preview where stitches were mapped to a pre-determined tileset. While this tile-based approach worked for knit, purl, and slip stitches, it could not represent stacked tuck stitches. As this is a crucial component of our target design space, we turned to models of yarn topology and found TopoKnit's algorithms described in enough detail that a JavaScript implementation was straightforward, which is included as part of our open-source contribution.

To draw the yarns in our swatch, the tiled stitch pattern designed in the repeat editor is used to produce a directed graph of yarn segments and their crossing points via Topoknit's approach, described above. The contact neighborhoods (yarn crossings) are initialized to positions in a grid based on their origin in the stitch pattern. Spline-based paths representing yarn segments are drawn between nodes using the method shown in Figure 4.10 to determine their start and end offsets from the contact neighborhood. Yarns are offset half of the current "yarn width"—the stroke width of the path. To correctly layer the paths, segments are drawn in the order they appear along the yarn. During drawing, they are separated into layers based on the operation which anchored the corresponding contact neighborhoods. For knit stitches, the legs are drawn on the front layer and the head and sinker segments are drawn on the back, and vice-versa for purl segments. Yarn segments traveling between knit and purl stitches are assigned to a third intermediate layer. To give the appearance of depth, the back layer is filtered to appear darker. KnitScape supports "flipping" the swatch to inspect the back of the fabric; this is achieved by reversing the stacking order of the stacking groups and re-applying the brightness filter.

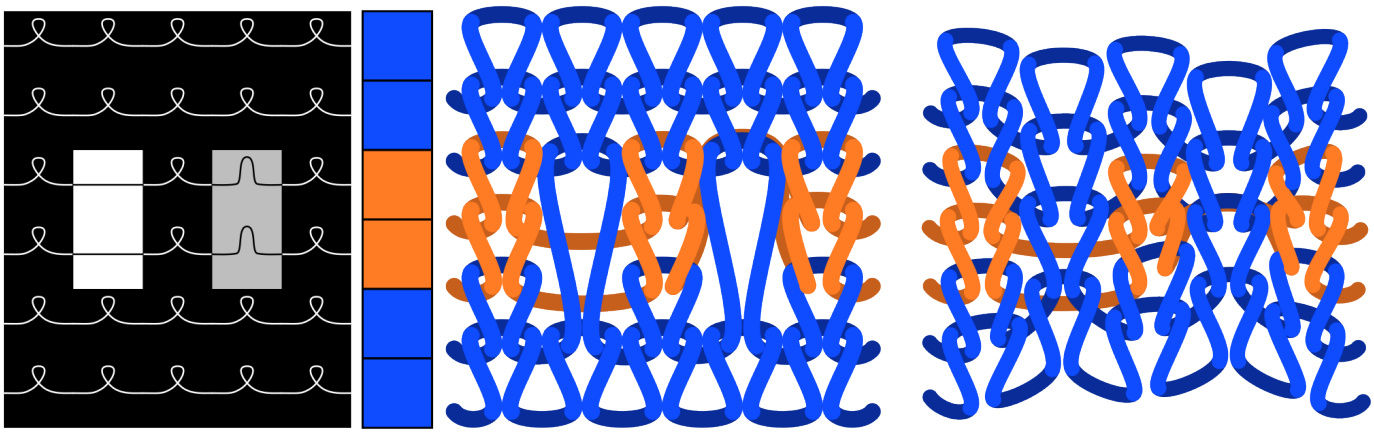


Figure 4.11 *A detail view of KnitScape's yarn relaxation. At left is the corresponding stitch pattern repeat and color sequence as designed in KnitScape. At center is the yarn view before relaxation. The non-relaxed view is quick to draw and constantly updates as the stitch pattern is edited. At right is the yarn view after the pattern designer runs the relaxation by clicking the "Relax" button. The legs of the slipped and tucked stitches have tried to contract to their rest length, pulling on the surrounding stitches. For this figure, we decreased the stroke width of the SVG paths to make it easier to follow the yarn paths.*

To simulate the deformation introduced by slipped and tucked stitches, we use the graph of yarn topology to build a two-dimensional spring system. Yarn segments are modeled as springs connecting two particles (the associated contact neighborhoods). We used different spring constants and target rest lengths for leg segments and head/sinker segments. Each yarn segment tries to contract to its rest length, applying a force between its two contact neighborhoods. As the simulation is run, the yarns are redrawn according to the updated positions of the contact neighborhoods. A detail view of a swatch before and after relaxation is shown in Figure 4.11.

Example Outputs

In this section, we demonstrate KnitScape's design space by knitting a series of example swatches. An overview of the swatches is shown in Figure 4.12, and they are each shown in more detail across Figures 4.13, 4.14, and 4.16. We sized the charts and simulations in these figures to depict multiple iterations of the repeat of interest; sticking to a common size would have been impractical as repeats vary substantially in size and aspect ratio. All patterns shown here are included in the hosted version of KnitScape in the pattern library.



Figure 4.12 *The fronts and backs of our demonstration swatches. All swatches were knit by the first author on 80 needles for 200 rows, with the same machine, settings, materials, and finishing process.*

Swatch Knitting Procedure

To enable comparison across swatches, all twelve were knit by the same person (the first author) following a standard procedure with the same materials (albeit different colors) and machine settings. We knit large swatches to capture multiple iterations of each repeat and account for differences in final height across patterns: 80 needles by 200 rows of knitting (plus 14 border rows in black yarn). Our standard procedure is described in Listing 5. We used Tamm's 3 Ply Astracryl yarn (100% acrylic) on a Silver Reed SK840 standard gauge flatbed machine. We also used the Silver Reed YC-6 color changer attachment, which automatically switches the yarn in the carriage when the carriage is moved all the way to the left of the needle bed. We chose to knit the swatches at stitch length seven as we knew from prior experience that it worked well for our yarn. However, stitch length affects the size and drape of the final fabric and some stitch patterns might be better suited for other stitch lengths if adapted into full pieces. Choosing the appropriate stitch length for a project is typically done by knitting the same pattern across a range.

Measurements taken from swatches are used to calculate stitch and row gauge, a necessary piece of information when knitting garments to size. Knitted fabrics also transform significantly through finishing and the normal washing process. This means it is crucial to process swatches via the same method that a finished piece would be treated before evaluating them. As our swatches were knit with acrylic (which is far more forgiving than natural fibers when it comes to washing and finishing), we

```
Set point cams to -40, 40 and push all needles in between to working position.
* If pattern includes any needles out of work, push them back to non-working position.
With carriage on right, e-wrap cast-on 80 stitches in black border yarn.
Set operation dial to knit.
Knit 7 rows, ending with carriage on left.
Set row counter to 000.
* Set operation dial to slip or tuck, depending on pattern.
Begin electronic patterning with end-needle selection enabled (to prevent
slip or tucks on the edge of the fabric)
* Knit in pattern until row counter reads 200.
End electronic patterning.
Change back to border yarn.
Set operation dial to knit.
Knit 7 rows, ending with carriage on right.
Bind off around gate pegs.
Cut yarn and pull through last loop.
```

Listing 5 *We followed a standard procedure when knitting each demonstration swatch. Asterisks indicate instructions which depend on the pattern being knit (e.g. whether to set the carriage to slip or tuck). On our digitally-controlled manual machine, the electronic controller selects which needles will perform slip and tuck operations automatically during knitting. The knitter must cast on, bind off, move the carriage back and forth, and press a button on the yarn change unit to switch yarns when desired.*

finished them by running them through a standard household washer and dryer cycle and applying steam to help them lie flat.

Changing the Stitch Pattern: Strategic Slip and Tuck Placement

Slip and tuck stitches "pull up" yarns from prior rows to hide the yarn that is currently being worked. In this subsection we describe how KnitScape can be used to investigate how slip and tuck stitches produce different effects on knitted fabric: sloped/curved lines, vertical and horizontal lines, connected areas of color, and recognizable motifs. Examples are shown in Figure 4.13.

Scrunched Ovals: Bending the grid

As knitted fabric is formed by many interlooped yarns; a slip or tuck stitch that pulls up a stitch from a prior row also pulls on the stitches around it. Our scrunched ovals swatch demonstrates how this can be used to coax a would-be horizontal line into a gentle slope, which is captured in the KnitScape simulation. While a negative of the

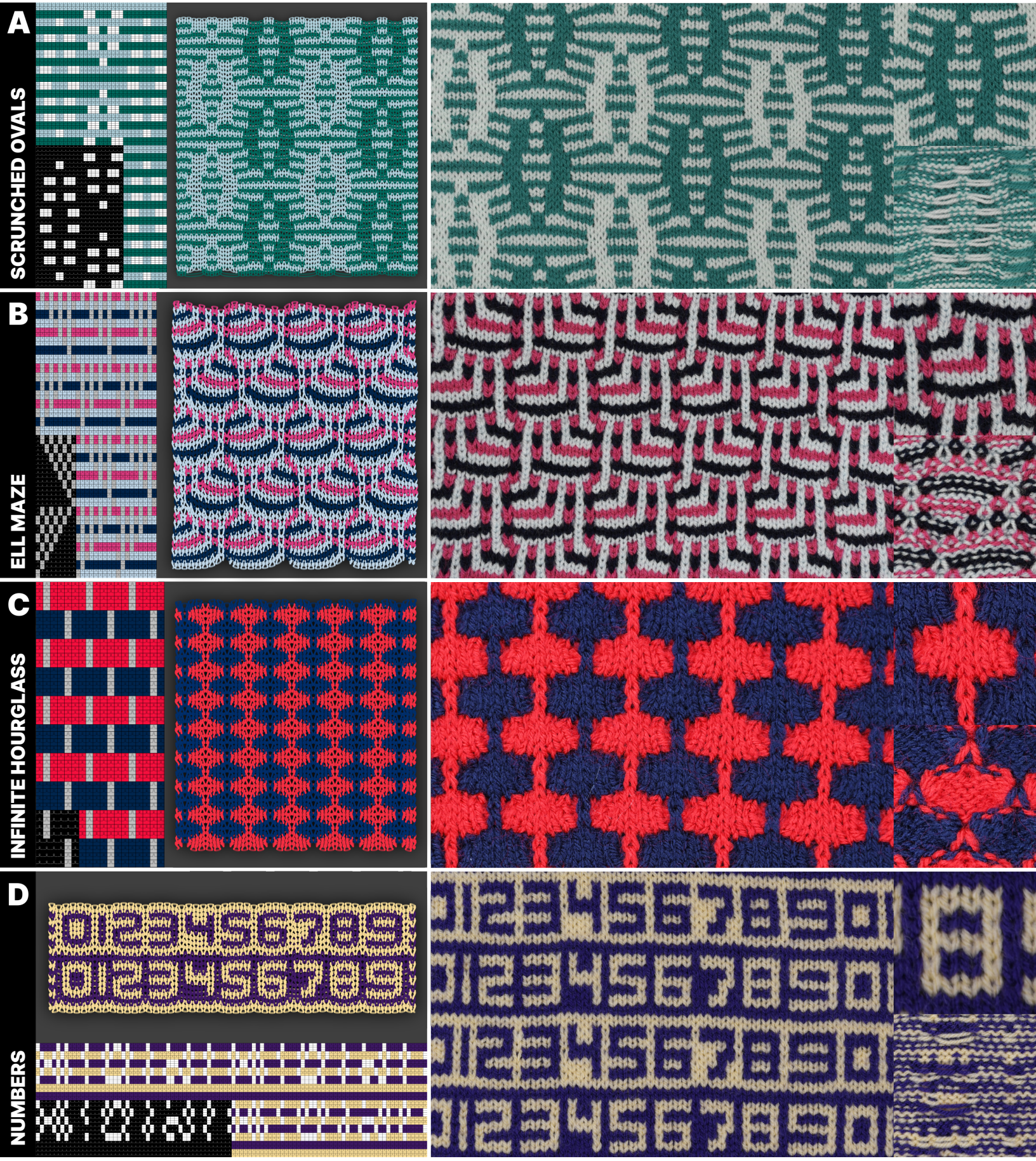


Figure 4.13 *KnitScape's stitch repeat (bottom left of colored operation chart, black and white) enables the strategic placement of slip and tuck stitches. The swatch simulation (middle) predicts knit outcomes (right).*

oval motif is somewhat recognizable in the operation repeat chart, the effect produced by the slip deformation would be very hard to predict without prior familiarity with this technique. Additionally, it would not be possible to achieve these gentle curves via fair isle colorwork, which is characterized by rastered/pixellated images.

Ell Maze: Combining horizontal and vertical striping

The Ell Maze swatch (a tuck pattern) demonstrates how horizontal and vertical lines can be used together in one design. Similar to Swatch A (Scrunched Ovals), this pattern would be poorly suited to fair isle colorwork as all horizontal bands of color would introduce long floats across the back of the fabric. The tuck stitches also introduce deformation, which gives the maze motif a shell-like appearance. The reverse side detail shows the tuck formation on the back of the fabric, which resemble small triangles.

Infinite Hourglass: Bridging larger areas of color

Stitches can be slipped or tucked over more than two rows. The limit depends on the machine and yarn used, but a general rule of thumb is to not stack more than six tuck stitches in a column [223]. This means that larger bands of color can also be manipulated with slip and tuck stitches. To demonstrate this, we knitted Swatch C, Infinite Hourglass, which uses a staggered sequence of four tucks to cross a four-row color band. The simulation captures how the colors patches are connected via the vertical tuck sequence, including the slight deformation introduced by the tucks pulling on lower rows. The reverse side detail shows how the tucks trap the red yarn behind the fabric.

Numbers: Quickly verifying a pattern result

Designing a slip or tuck pattern for a recognizable form (such as a number or character) may be time consuming without a way to verify that the correct color effect will be produced when knitted. The KnitScape simulation enables quick verification that a slip/tuck repeat will produce the desired pattern when knit, enabling trial-and-error to take place at design time without wasting materials. We designed a slip stitch pattern for numbers zero through nine in KnitScape and knitted Swatch D. Direct manipulation of the operation repeat enabled us to place slips to stay within the four-stitch constraint. The reverse side detail shows the few minor slips along the back of the

fabric, the largest occurring due to the negative space around the four.

Planning the Color Sequence

In KnitScape, a color sequence is independent from the stitch pattern. Tuning these two parameters together is powerful, while manipulating them separately can lead to surprising results. In this subsection, we present four swatches (shown in Figure 4.14) we knitted to demonstrate different color-sequence design strategies.

Robovision: Extending color sequences beyond the stitch pattern

As colors are planned separately from the operation repeat, a small operation repeat can be used across a larger sequence of color changes. The KnitScape chart preview tiles both the operation repeat and color changes, regardless of their sizes. Swatch E, Robovision, demonstrates the effect of using the same operation repeat across different colorways.

Squarget: Adding More Colors per Row

Carefully planning the placement of slips enables multiple colors to appear in one row. In Swatch F, Squarget, the carriage must make six passes to achieve three colors in one row. The effect is visually striking when viewed from the front, but the large number of slips in this pattern may limit its use. However, there are a variety of ways to mitigate this (e.g. by crocheting the slips together during knitting).

Pseudoweave: Designing a Mock Woven Effect

Many slip and tuck patterns have interesting effects produced on their “wrong” sides. In Swatch G, a large number of slips produces a horizontal band of yarns that stretches across the back of the fabric. When these bands are each given a color, they resemble a ribbon that is interwoven with the cream-colored yarn which runs vertically. At first glance, it may be difficult to tell that this is a knit fabric unless inspected closely. This pattern is one that may benefit from a looser tension, as the resulting knit was quite tight.

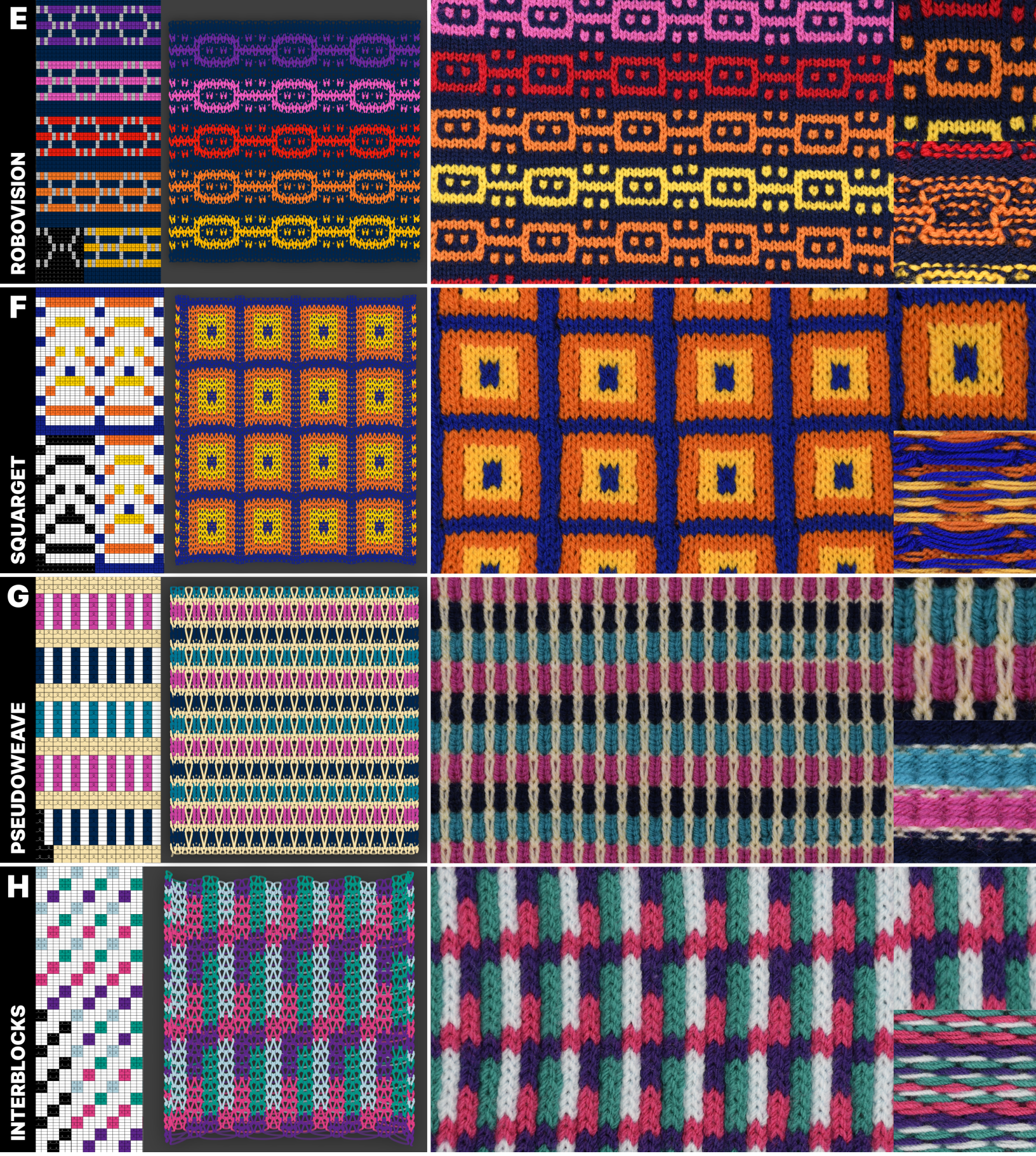


Figure 4.14 *KnitScape's color sequence editor is separate from its stitch editor. Used together they can produce surprising results. Compare the colored operation chart (left) to the simulation (middle) and knit swatches (right).*

Interblocks: Navigating Unintuitive Patterns

Some of our operation charts (e.g. for Swatches A and F) bear a passing resemblance with the knitted result. However, predicting this relationship gets progressively difficult with more complex color sequences and stitch patterns. For Swatch H, we wanted to knit a four-color pattern that resembled building blocks. If we were designing solely based on our colored chart, it would have been difficult to tell if the color sequence properly connected the blocks of color. KnitScape's simulation made it is easy to verify that our color assignments were producing the intended effect and not spend time knitting swatches with an unclear outcome. This fabric was the smallest (in terms of width and height) of all swatches we knitted due to the large number of floats across the back. Despite this, this pattern does not exceed the recommended four-stitch float maximum.

Removing needles from work

In knitted open-work, openings or holes in the fabric are produced when adjacent stitches are not fully joined [223]. There are many ways to knit open-work; on a knitting machine, one way is to leave a needle empty, either by transferring a prior loop to another needle or never knitting on it in the first place. Open-work can also be produced by dropping a stitch and causing the stitches below it to unravel, a common hand-knitting technique which can also be carried out on a machine. In KnitScape, the needle editor can be used to design a repeat sequence that defines which needles are in and out of work. In the preview chart, the columns above non-working needles are grayed out to indicate that no operation will take place.

On a manual knitting machine such as the Silver Reed SK840 we used to knit our demo swatches, needles are removed from work by pushing them back to a position where the carriage passes over them without actuating them (Figure 4.15). The yarn therefore passes in front of the needle's position without forming a loop, effectively doubling the distance between two adjacent stitches (Swatch I). Placing tuck stitches alongside a ladder can selectively bunch together the ladder "rungs" via a technique known as "tuck lace" (Swatch J). Combining this with a color change sequence enables further opportunities to hide and show color (Swatches K and L).



Figure 4.15 *Needles are removed from work by pushing them to a position where they are not actuated by the carriage.*

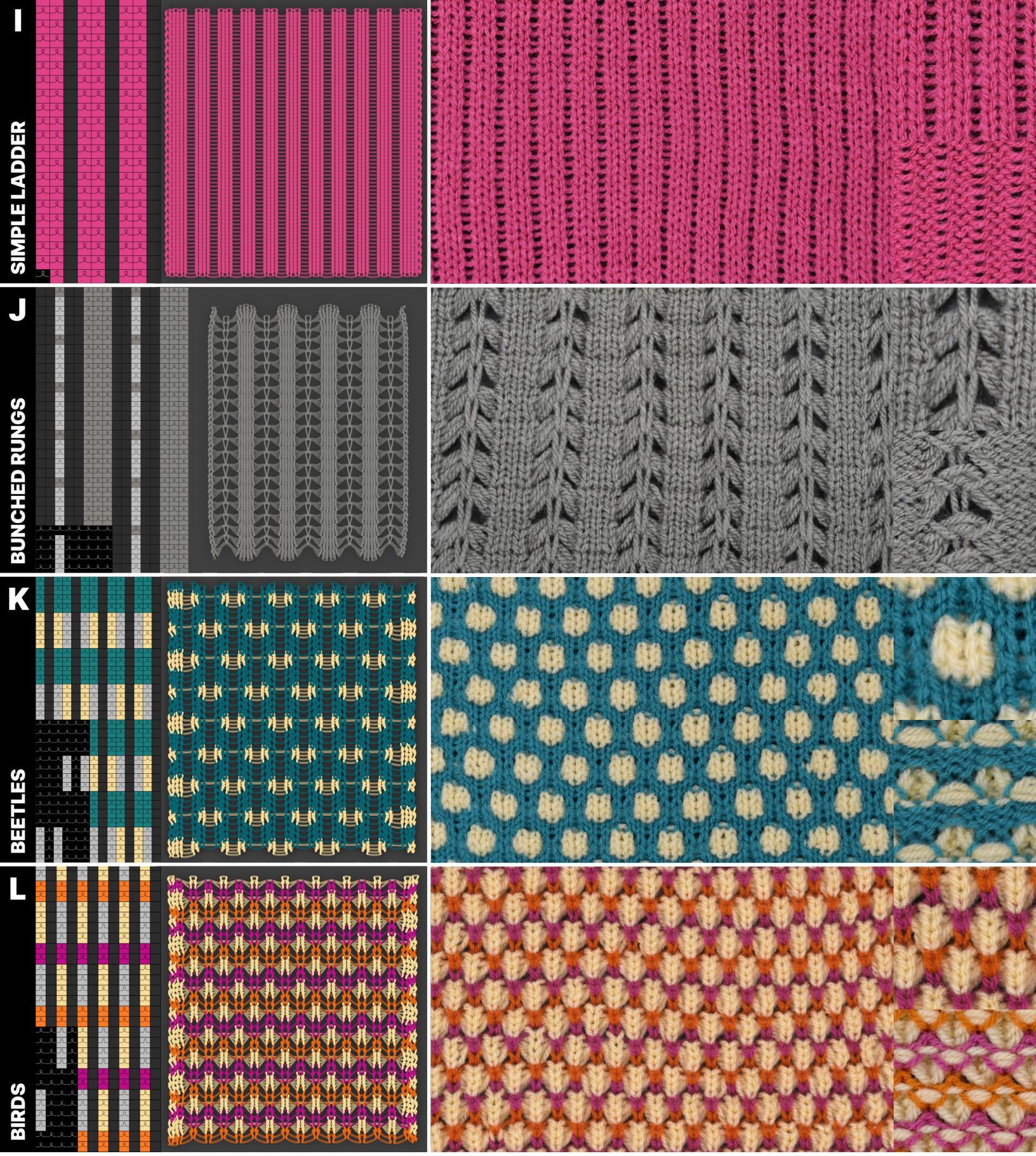


Figure 4.16 *Needles left "out of work" form ladder structures as yarn passes in front of them without forming a loop. KnitScape's needle position editor enables design of ladder-based openwork. Needles not in work are represented by a column of grayed-out stitches in the pattern chart.*

Simple ladder: Introducing open spaces

To demonstrate the effect of removing a single needle from work, we knitted a simple ladder swatch (Swatch I). Two needles are in work for every needle left out of work, and there are no other patterning or color changes. Yarn passes in front of the empty needles to leave a "rung" which connects the legs of adjacent stitches, which is captured in the KnitScape simulation. Small holes are visible between ladder rungs on both the front and back of the knitted swatch.

Bunched Rungs: Manipulating ladder rungs with tuck stitches

Tuck lace is a deceptively simple open-work technique where the "rungs" of a ladder structure can be "bunched" together by holding them with tuck stitches on either side (Swatch J). This exaggerates open spaces in the ladder and dramatically elongates the held stitch, which can also easily take up slack from the empty rungs. In Swatch J, a vertical sequence of tuck stitches is bordered by two empty needles on each side. The KnitScape simulation captures the bunching and dramatic elongation of the tucked stitch.

Beetles: Hiding and showing color

Tuck lace offer another opportunity to hide and show colors by enabling yarns hidden by tuck stitches spend more time at the back of the fabric. Normally on knitting machines without sinkers (a component which stabilizes loops that are on needles), a tuck stitch must be bounded on both sides by a knit or purl stitch to prevent it from falling off when the needle is actuated. However, this is not necessary when separating two tucks with an empty needle (Swatch K). This means that we can design tuck colorwork where multiple tucks seem to appear in a row. Ladders bounded by tucked stitches are noticeably smaller than those bounded by knit stitches. The tucks neatly bundle the light yellow yarn on the back of the fabric.

Birds: Manipulating color with texture

Tuck lace fabrics are significantly textured, which offers another opportunity to strategically place color for different effects (Swatch L). The KnitScape simulation shows how the pink and orange yarns are pulled together via stacked tucks, producing an effect of criss-crossed diagonal color. The columns of light-yellow knit stitches pop out

to produce bumpy waves, which obscure some of the lines of orange and pink. On the back side of the fabric, the tucked light yellow yarn peeks through the purl side of the pink and orange stitches.

Discussion

Knitting is a complex material process with an immense possibility space, which makes it a fascinating topic of research to us (and, we suspect, many other researchers). However, it also presents a practical challenge when choosing how to black-box aspects of the knitting process in new design tools. KnitScape uses a chart-based pattern format, an abstraction which is typically considered to be low-level (and certainly may become an unwieldy design representation if used to visualize every stitch in a finished piece). In our experience using KnitScape, we felt that the smaller scope of a chart-based repeat made it an effective input parameter for exploring the design space of slip and tuck colorwork. With KnitScape's yarn simulation providing immediate feedback when editing the chart, we learned to anticipate *how* a tuck or slip stitch would interact with the surrounding stitches. With practice, we were able to use this material intuition to design patterns such as our Numbers swatch in Figure 4.13.

There is a clear opportunity for a future tool to explore the "inverse problem" in slip and tuck colorwork, i.e., determining the slip and tuck placement and color sequence which will approximate an input curve or image. This would enable goal-driven design of slip and tuck colorwork patterns, preventing some low-level work but requiring a designer to have a well-defined idea of a desired output pattern. We think that knowing ahead of time which patterns will "work well" using this technique will be difficult without first understanding the underlying material interaction. With this in mind, we think that there are many cases where such a tool would be powerful, but that it would not be a replacement for KnitScape's emphasis on gaining material intuition via exploration and discovery of new patterns.

This is in line with findings from prior studies of creative practice, which found that practitioners opt for lightweight and flexible tools when iterating with materials [244] and that over-emphasis on automating aspects of the creative process can discourage

them from adopting and using design tools [137]. We argue that KnitScape heeds these recommendations, adopting methods that reflect how knitters mix and match techniques and design elements when designing knitted pieces. We found our prior understanding of both slip/tuck colorwork and shaping to be useful when knitting the shirt shown in Figure 4.1, as understanding them separately helped us anticipate how they might interact. We therefore think that tools should support designers in building ground-up intuition about processes and how (digital) designs will play out in physical material. This is in line with others who argue that design tools for physical making should support open-ended exploration of material constraints and process prototyping [11, 125, 85], for example by creating lightweight and highly specific tools. Such tools have been made for brioche knitting [9], clay carving [235], punch needle embroidery [81, 43], quilting [131], and woodwork [135, 239].

We are very excited to continue working on KnitScape and see many possible extensions. Future improvements to the tool include extending it to support additional families of knitting techniques, such as lace or texture patterns. We have already begun working on extending the KnitScape simulation to 3D in order to better represent highly three dimensional texture. Swatches K and L, shown in Figure 4.16, have more depth than the KnitScape simulation is able to capture. KnitScape can currently be used to visualize the yarn topology of short rows (which can be represented via horizontal rows of slips), but the current 2.5D preview does not show how this would deform the fabric into a 3D surface.

We think tools which prioritize exploration of emergent material effects will be especially important for researchers looking to incorporate dynamic or "smart" materials into future soft interfaces. We are interested in how KnitScape's current simulation could be extended to model dynamic materials such as shape-changing fibers [61, 118]. KnitScape's focus on exploration and discovery of emergent material effects could be valuable to researchers who are beginning to investigate new applications for these materials. More broadly, we argue that openness and interoperability are key, as this enables the development of workflows that span multiple tools. Beyond opening black boxes, this means playfully connecting workflows across materials and environments [134, 72]. For knitting, research is already underway towards formalizing designs and establishing exchange formats, especially for fully-automated knitting [140, 87]. However, much future work remains for HCI researchers developing robust

infrastructure that supports connections across domains.

Conclusion

We presented KnitScape, a browser-based tool for designing and simulating knitted slip and tuck colorwork swatches. In KnitScape, knitters can specify the stitch pattern: a repeating group of stitches such as knit, purl, slip, and tuck; the color sequence: which colors to use in which row; and if needles of knitting machines are to be in or out of work for knitting tuck lace. To simulate the knit swatches, we implement a client-side Javascript yarn-level representation based on TopoKnit, assigning spring values to the connections between yarn crossing points in the textile. The simulation is rendered as SVG; this lightweight implementation means that knitters do not need computational resources to see how their textiles will behave. To demonstrate the design space of colorwork with KnitScape, we machine knit twelve swatches that showcase a variety of stitch patterns, color sequences, and lacework. These showed how KnitScape enables quick iteration and exploration of interdependent parameters in colorwork knitting.

Chapter 5

KnitScape II

Abstracting Knitting Design Elements with Blended Raster/Vector Primitives

This chapter expands on the initial version of KnitScape presented in the previous chapter, exploring the question: once someone has developed material-level intuition (e.g., by using the previous iteration of KnitScape to iterate on colorwork patterns), how can they apply that intuition to the composition of more complex designs? This was done by adding composable primitive elements to the KnitScape interface, enabling groups of raster operations to be manipulated with vector elements. With composable primitives, insights about emergent material behavior (such as the tendency of certain stitch patterns to produce fabric that curls) can be incorporated into more complex designs (such as self-pleating patterns). The content of this chapter is adapted from a paper published at UIST 2024 [245].

Introduction

Knitting enables the fabrication of soft objects through a set of discrete operations on a yarn or set of yarns [223]. By combining different stitches and yarns, we can manufacture complex, multi-material, three-dimensional shapes, with varying micro and macro material behaviors [7, 144, 111, 118]. Color patterning (also known as colorwork) and texture effects such as lace, cabling, and ribbing are extensively used in the design of knitwear and can introduce aesthetic and functional attributes to knits [9, 246], including the integration of elements of great interest to HCI researchers such as sen-

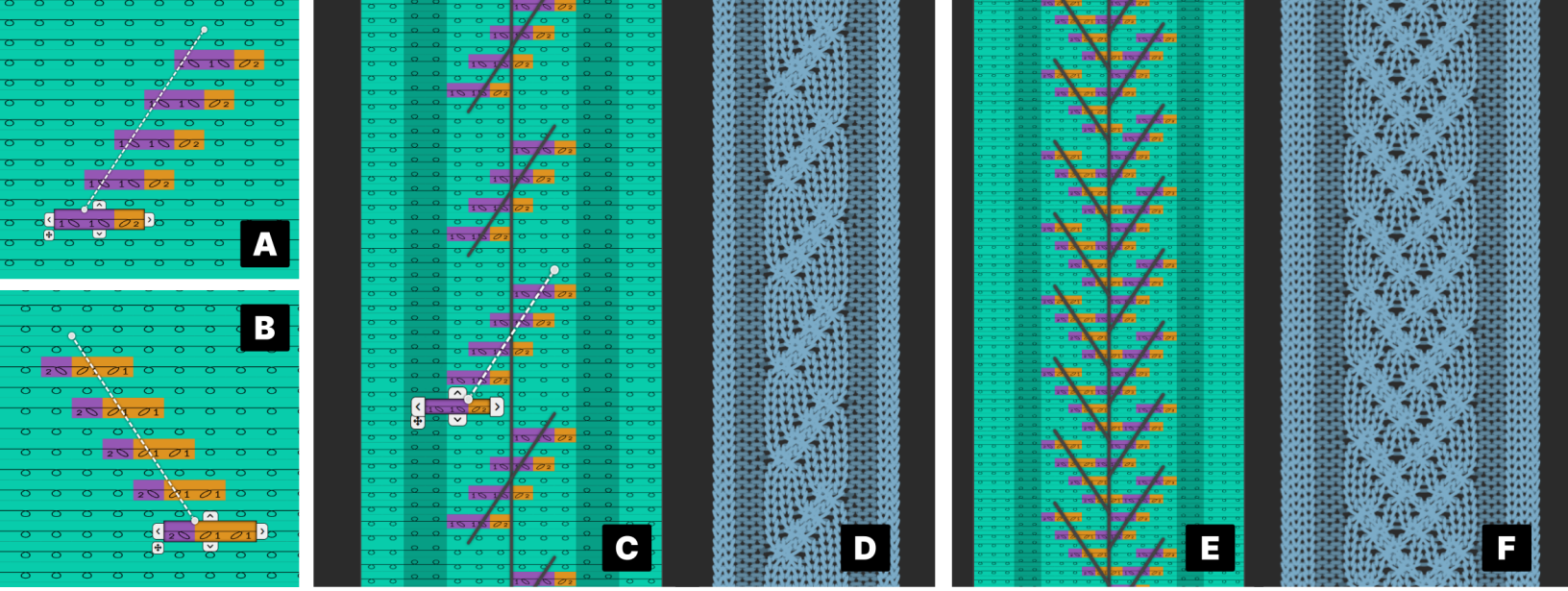


Figure 5.1 *In our design environment for editing knitting charts with blended primitives, a chart of knitting instructions is rasterized from layered vector boundaries and paths with associated raster stitch and yarn blocks. These cabled twist and braid patterns are designed using stitch paths we defined to encode a "right-leaning twist" (A) and a "left-leaning twist" (B). By placing the right twist along a stitch path that defines a purl border (C), we can produce a twist texture (D). By staggering both the left and right twists (E), we can produce a braid texture (F).*

sors and shape-changing components [180, 147, 61, 148]. Knitwear designers have common strategies for producing these attributes, but need to ultimately specify their knits with low-level descriptions. The lack of appropriate design representations can result in design challenges, as changing patterns stitch-by-stitch is labor intensive.

For example, a *cable* is a surface texture which looks like a raised column of interwoven or twisted fabric [120]. Cables are a core element of Aran sweaters (such as the purple sweater in Figure 5.2), where many cables adorn on the surface of a knit to create intricate patterns. Cable knitting involves changing the order of certain loops at regular intervals. A designer has many variables to play with when designing cable structures, including the number of loops at each crossing, the crossing direction, crossing frequency, the column lean, etc., and can additionally incorporate colorwork and texture techniques to produce stunning surface patterns [76]. All of these variables can have a dramatic impact on cable appearance, yet their interdependence defies top-down abstraction: to design new cable structures, you must specify them on the individual stitch level, always reconciling cable-level design decisions with other factors such as shaping.

The problem of design representations in knitting is familiar to HCI and graphics researchers, who have formalized low-level knitting operations in order to develop



Figure 5.2 *Examples of knit surface patterning effects on a selection of sweaters. As opposed to overall fabric topology (e.g., a sleeve) or the individual stitch, these* mid-level effects *[9] determine the texture and color of knits.*

higher-level primitives that can be used to design new knit objects [152]. This body of prior work has largely focused on 3D shaping-oriented primitives which represent the overall surface topology of knit objects [152, 115, 106, 112, 170, 169]. In contrast, the work presented here seeks to support design tasks which require a different level of abstraction than overall surface topology, namely the design of knit surface patterning effects (i.e., *mid-level effects* [9], examples of which are shown in Figure 5.2). What kinds of representations might we need to design new cables, lace, colorwork, and texture patterns?

As another example, *intarsia* is a colorwork technique commonly used for patterns with large blocks of color (such as the corn and crow sweater in Figure 5.2). While it may appear that intarsia patterns would be relatively easy to design in, say, a pixel-based paint tool, there are knitting constraints that a designer must consider during the design of an intarsia pattern, including how yarns of different colors are joined (to prevent gaps between yarns), the direction each yarn is moving in a row, the slope of the boundary between yarns (a slope that is too gentle may lead to long floats), and on which side the yarn starts and ends. An example of the impact of these constraints is provided in subsection 5. Designing intarsia patterns often involves resolving these constraints via decisions which 1) must happen at a low level, and 2) impact the final appearance of the pattern, sometimes dramatically. For example, this may include something as simple as resizing the height of a motif so that it totals an even number of rows, ensuring a yarn will return to the same side it started on.

As a final motivating example, consider a designer who wants to create different amounts of stretch in different regions of a garment, for example: a sweater with conformal cuffs, a warming densely knit body, an airier knit for the sleeves, and tough patches for the elbows. How can they prototype these different textures? While some textures are well-established, such as a ribbing for cuffs, we have only scratched the surface of the mid-level material effects that are possible. Recent work has explored how mid-level effects can, for example, be tailored to produce material properties ideal for particular regions of garments [144]. How different stitches and yarns impact the final textures in knits is difficult to understand based on low-level descriptions alone, and rapid iteration on prototypes will be crucial as we look to understand the possible material effects of novel functional fibers (e.g., [61, 4]).

Better design representations would enable higher-level manipulation of surface-level knit effects. However, developing abstractions for knitting is non-trivial. First of all, knitting design choices defy a strict hierarchy: stitches and yarns are interdependent and both impact elements of the resulting textile. Second, knits are subject to strict set of fabrication constraints, which will change depending on knit pattern design choices (e.g., required number of transfers in a row), yarn selection (e.g., how stretchy or strong a particular yarn is), machine features (e.g., number of yarn feeds), and other implementation details. Therefore, our guiding research question is: How can we build design representations for knitting elements that support higher-level direct manipulation while maintaining access to stitch-level specification?

To address this research challenge, we contribute *blended primitives* for abstracting knit design elements. Specifically, we blend raster-based stitch and yarn blocks with vector-based paths and boundaries. This allows us to specify and manipulate the relationships between design elements, translated to groups of low-level knitting operations, and use these representations to generate machine knitting instructions. For example, we can design a raster-based stitch block that creates a lace texture, then repeat that stitch block in a region of a garment as defined by a vector-delineated boundary to create a lace patch. By using techniques from computer graphics, we can determine reusable high-level strategies for stitch block repeats and other rasterization challenges.

To demonstrate how blended primitives support the exploration of a knit design space, we contribute an open-source browser-based design environment. Links to the open-source code repository and hosted site can be found on the project page: https://depts.washington.edu/machines/projects/blended-primitives/. Our visual software supports direct manipulation of knit design patterns, including repeating stitch blocks (which can include knit, purl, tuck, miss, and transfer operations), yarn blocks (for techniques such as fair isle colorwork), shaped knits (specified through increases and decreases boundary conditions or short rows), and transfer operations (for designing cables and other knit textures). To demonstrate our design tool, we have knit several examples that showcase different aspects of the design space.

In summary, this chapter contributes:

- Blended primitives for designing mid-level knit effects: stitch paths, yarn paths,

stitch fills, and yarn fills.

- An open-source tool for designing with blended primitives, consisting of a chart-based editing environment and yarn-level simulation.
- Examples of mid-level abstractions for common knit design elements, including intarsia and fair-isle colorwork, lace, and cables.
- Two domain demonstrations: intarsia-based sensing structures and origami-inspired pleat patterns.

We conclude with a discussion of opportunities for HCI systems researchers in design tools for knit textiles and what we see as promising areas of future work.

Related Work

Machine knitting has recently seen a surge of attention from HCI researchers who recognize it as an essential technique for the production of e-textiles, soft interfaces, and functional fabrics, as well as an established outlet for personal expression. In this section we discuss how our contributions fit into the landscape of prior work on knitting design representations and associated tools. Additionally, we review work on rasterization and fabric modeling and simulation which is relevant to the implementation of our design tool. While we focus on machine knitting in this chapter, we recognize that interest in textile fabrication also extends to other domains, including weaving [66, 49] and embroidery [70, 71, 78].

Design Representations for Knit Objects

There appears to be a general consensus among researchers that the complexity of knit programming, and the associated lack of tractable design primitives, poses a barrier to fully exploring the extensive design potential of machine knitting. Prior work has looked to address this challenge, largely through development of high-level shaping primitives which can be used to represent the overall surface topology of knit objects while respecting the fabrication constraints of the knitting process. This includes generalized tubes and sheets [152], which was later extended to include splits

and merges and deployed in the context of an interactive design tool [112, 115]. Other representations include stitch meshes [258, 169, 170] and coarse template meshes [106].

We distinguish our work from prior approaches by emphasising our focus on supporting the design of *fabric-level structures* as opposed to high-level 3D shape. High-level shaping primitives largely divorce three-dimensional form from the underlying stitch-level descriptions; this is advantageous for design tasks which are concerned solely with global fabric topology (such as constructing a garment silhouette). However, we argue that the task of designing novel functional and aesthetic fabric-level structures requires fine control over individual loop positions. For example, prior work has demonstrated that small changes to loop positions can have a dramatic impact on the appearance of slip and tuck colorwork patterns [246] and the performance of knit sensors [4, 14]. We therefore think it is crucial to maintain direct access to stitch-level manipulation when designing fabric level effects, which our blended primitives approach supports by enabling this control via raster-based blocks containing stitch-level operations and yarn assignments. Prior work has explored different approaches to surface patterning largely through the development of textual domain-specific programming languages [112, 88, 87]; our work instead presents a visual design environment where operation positions can be controlled via direct manipulation of associated vector paths and boundaries. This is more in line with the approach described by [9], which is specific to brioche knitting.

In parallel with work on design representations, researchers have worked to build out the infrastructure which will be essential for future development of robust machine knitting workflows. This includes process-oriented models of yarn topology [109, 110], domain-specific programming languages [89, 87], compilers [152, 141], algorithms for scheduling and transfer planning [159, 141, 139, 263, 112], and debugging-oriented visualizations [263]. While the work we present in this chapter is focused less on infrastructure and more on design representations, we found this prior work to be a valuable reference during implementation of our design tool.

Fabric Modeling, Simulation, and Rendering

Broadly, yarn simulation has been an area of intense interest in HCI, computer graphics, and textiles communities [129, 36, 107, 108, 264]. Early work achieved draping behavior in knitted fabrics [22], and proposed animation methods with particle systems [156] and splines with animated control points to model the micro-structure of knitted fabrics [211]. Kaldor et al. [107, 108] published the first system capable of simulating entire garments at the yarn level, with computation of contact response of yarn-yarn collisions. Yuksel et al. [264] introduced the stitch mesh representation that incorporated mesh-based and yarn-level relaxation. Further work has focused on efficient implementations through nodal discretizations [190], combined representations to retain large-scale behaviors of cloth [32], and realistic rendering of fabric appearance [33, 268]. Recent work continues to demonstrate new ways in which yarn-level mechanics can improve fabric simulations, such as using implicit contact handling to simulate multi-layer fabric structures [213], animating yarn-level geometry on top of a deforming mesh [224], and inverse-modeling yarn-level mechanics from real-world measurements [225]. For a more extensive review of simulation background we refer to Castillo et al. [33] and Casafranca et al. [32].

A challenge with relying on real-time yarn-level simulation as part of a design workflow is that computational intensity rapidly increases as pieces get larger. Leaf et al. [129] offer yarn-level simulation of swatches at interactive rates, but require GPU implementation. Our system extends the lightweight browser-based simulation introduced in KnitScape [246]. A topology graph [109] is combined with a particle-spring simulation to visualize local deformations in knitted swatches. In our work we demonstrate how mid-level effects like color and texture patterns offer an ideal opportunity to deploy yarn-level simulation in the context of an interactive design tool.

Rasterization

We draw on foundational concepts in computer graphics for rasterizing geometric primitives and apply them to this domain of knit chart editing. Line drawing algorithms [24, 189, 249] and scan-line polygon filling [82] were among the earliest graphics methods for efficiently approximating exact mathematical representations of geom-

etry as pixel positions in the frame buffer. Analogously to rasterization, knitting chart design entails converting geometric lines and regions into discrete stitch positions. Many desired properties are also shared. For example, in origami pleat patterns (Figure 5.16) the fold lines should be uniformly one stitch wide and avoid artifacts such as gaps when projected onto a knitting chart for consistent folding behavior and appearance.

Pixel art has maintained prominence in computer graphics with popular editors (e.g., Aseprite and Pixelorama), and algorithms to facilitate pixel art creation through image abstraction [69] or rasterizing vector line art with minimal artifacts [96, 97]. Some community-built tools aid in the translation of images to a format readable by manual knitting machines (e.g., img2track [42] and AYAB [13]), but these are focused on translation of existing patterns to a knittable format rather than providing a full design environment. In contrast, our system integrates editor features such as layers and tilemaps into charting-specific workflows.

Blended Primitives

Knitting patterns are typically designed in gridded charting environments where individual operations are assigned cell-by-cell and yarns are assigned row-by-row, similar to a pixel-based painting workflow. While raster charts are a clear and widely used notation, they are not conducive to higher-level manipulation of groups of instructions beyond tile-based selection and repetition. Additionally, many kinds of raster edits can be destructive (such as resizing workspace boundaries or translating stitches) and/or highly manual and repetitive (such as changing the slope of a line). Vector-based editing could potentially solve many of these problems, however, many knitting design elements (such as base texture or colorwork patterns) map well to a raster or grid-based presentation. Therefore, our goal is to extend a raster-based charting environment with the ease of vector-based editing.

We propose **blended primitives** for knit pattern design: vector paths and boundaries which can be used to manipulate raster blocks of stitch instructions and yarn assignments. In our design environment, blended primitives are layered and raster-

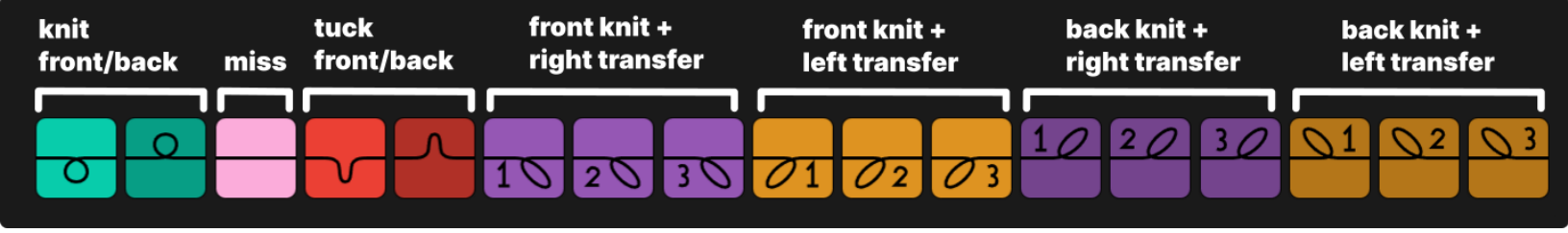


Figure 5.3 *Knitting operations: The charting language in the blended primitives editing environment supports knit, miss, tuck, and composite transfer operations.*

ized to create two charts which are overlaid to form a compact chart view: one which encodes yarn assignments and one which encodes stitch instructions. To enable this compact view where multiple yarns can appear in the same row of instructions (as opposed to most editors where only one yarn can appear per row), our system provides automatic yarn separation for fair isle and intarsia patterns, described in Section 5.

Maintaining quick and easy access to low-level stitch and yarn assignments is one of the core goals of blended primitives. In our editor, modifications to the yarn and stitch charts can occur either through direct manipulation of vector elements (e.g. by dragging path segments and control points or changing their stacking order) or via raster edits to their associated stitch and yarn blocks. In the rest of this section, we describe the pattern elements used to design knits in our system, give an overview of our tool interface, and provide a walkthrough of an intarsia design workflow in our system. In following sections, we show how blended primitives can be used to design common knitting elements through a series of demonstration examples, including intarsia and fair isle colorwork, lace, and cables. Later, we show two domain demonstrations in which blended primitives are used to design origami-inspired pleat patterns and intarsia-based sensing structures.

Pattern Elements

In our design interface, patterns are built from combinations of three kinds of primitive elements: blocks, boundaries, and paths.

Blocks

A *block* comprises two bitmaps (one defining yarn assignments and one defining operations) which share a common origin at their lower left corner (Figure 5.4). All boundaries and paths have an associated block which is used during chart rasterization.. Additionally, “free blocks” can be used as a one-off non-destructive override for yarn and stitch instructions on lower layers, which is useful when designing an indi-

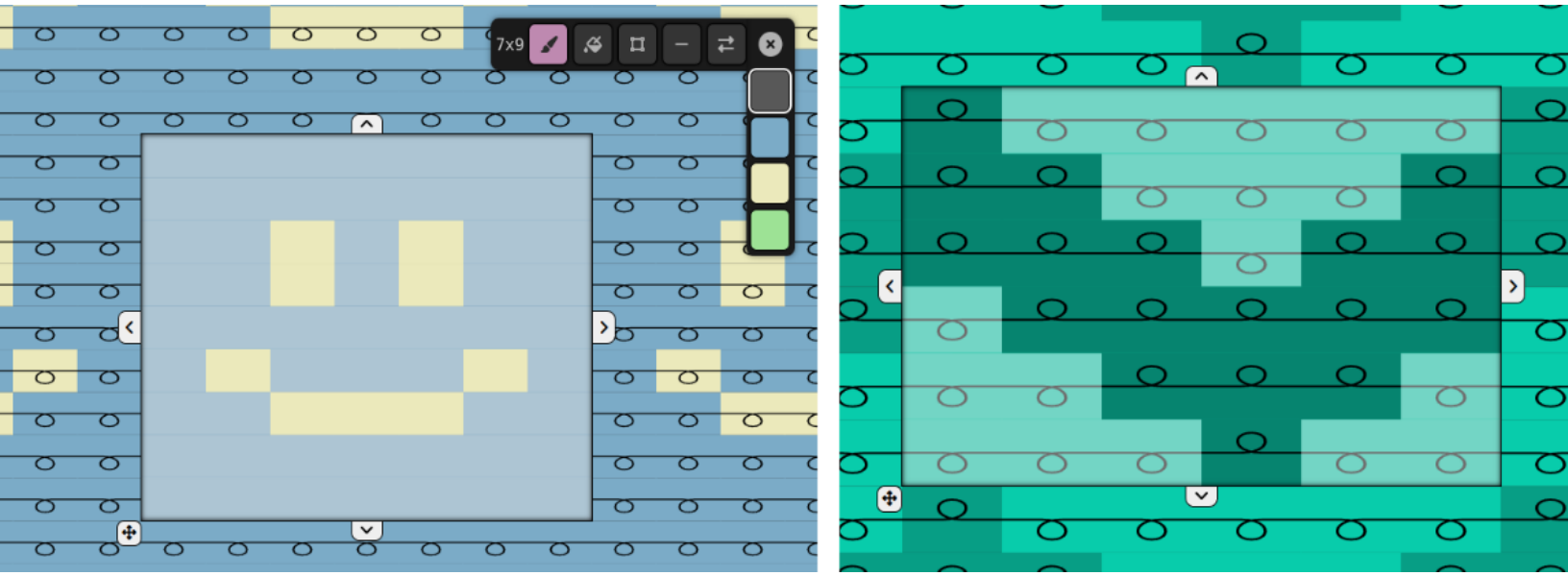


Figure 5.4 *Blocks are raster bitmaps. Yarn blocks (top) encode yarn assignments, and can be edited with yarns from the current yarn palette using pixel editing tools. Similarly, stitch blocks (bottom) can be edited with operations from our supported operation set. Blocks can be quickly resized by dragging the arrows at their edges. The block origin can be easily repositioned using the "move" dragger at the bottom left corner.*

vidual colorwork or texture motif. For blocks associated with boundaries and paths, the block origin is defined as an offset to the first point in boundary or path, which can be moved to change the position of block repetitions. For free blocks, the block origin is an offset to the global pattern origin.

Stitch blocks are bitmaps which include supported operations (described in Figure 5.3) as well as the "transparent" stitch, which inherits the stitch assignment of a lower layer. Yarn blocks are bitmaps which include indices of yarns in the global yarn palette. Similar to stitch blocks, yarn blocks can also include a "transparent" yarn which inherits a yarn assignment from a lower layer. A new block contains a 1x1 bitmap with a transparent stitch and a 1x1 bitmap of a transparent yarn.

Boundaries

A boundary is a closed polyline path which outlines a region in a chart to be filled with a stitch and/or yarn pattern (Figure 5.5). Figure 5.13 shows how boundaries can be used to position colorwork motifs and a rib texture for a simple doodle cowl. Boundary edits are not destructive to their stitch and yarn fills, making it easy to use them to lay out and reposition different patterning blocks. This means that patterns like the cowl shown in Figure 5.13 are very easy to modify, e.g., by dragging the boundary edges. The tiling origin of a stitch and yarn block can be moved by dragging the arrows at the bottom left corner of the block. This is used to align blocks relative to their boundaries or other blocks, e.g., to avoid strange repeat cutoffs at the edges or stagger different motifs evenly.

Paths

A path is a polyline along which a stitch and/or yarn block is tiled. Like boundaries, they are easy to manipulate by dragging control points and segments, adding and removing points, and reordering layers. Path appearance is easily changed by editing the associated block (Figure 5.6). We support four different tiling modes to control block position along the path (described in 5). The block can also be offset from the associated path using the "move" dragger at its bottom left corner. This can be used, for example, to change which side of a path a single-cell wide block appears on.

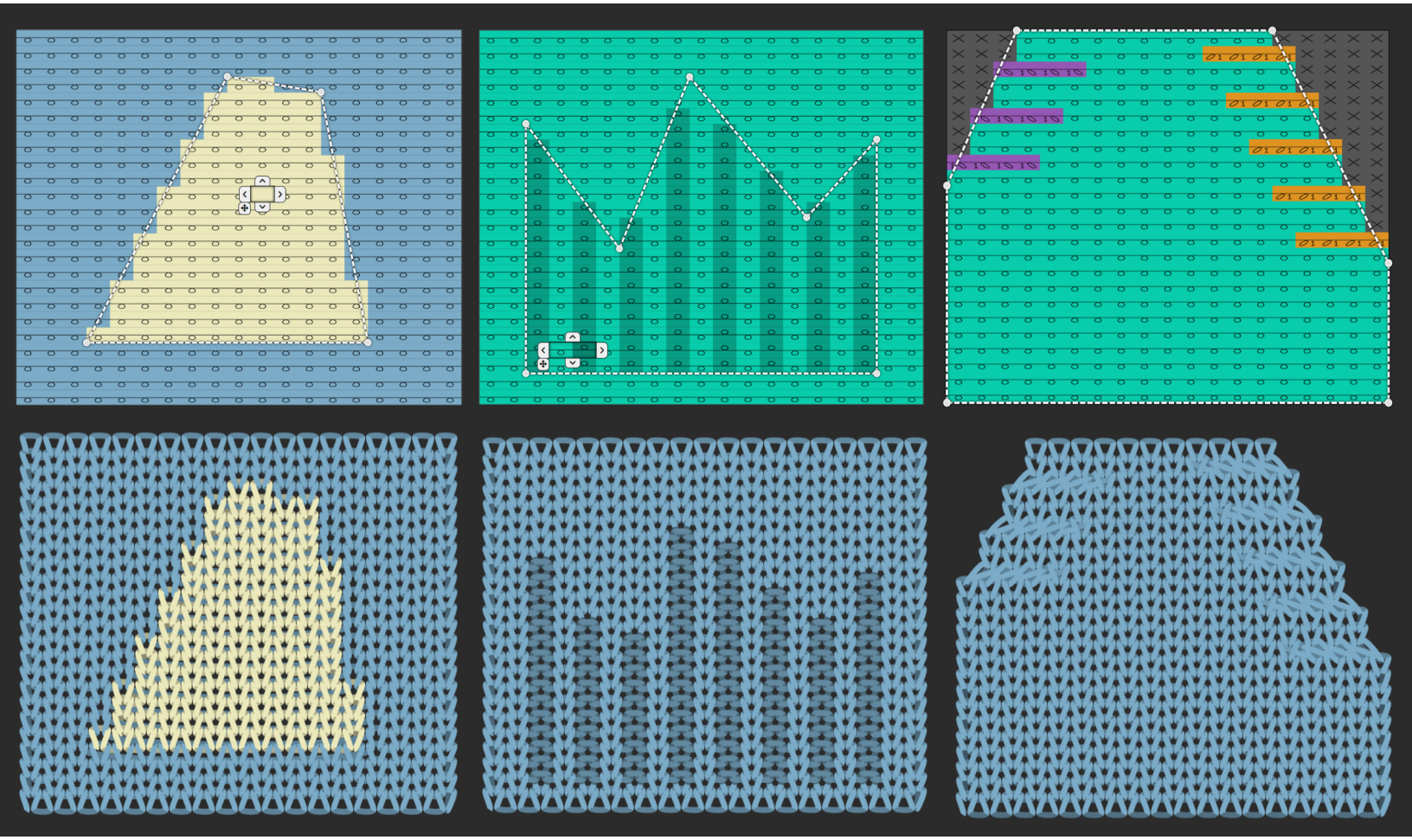


Figure 5.5 *Boundaries can be used to assign yarns (left) and stitch patterns (center) to regions of a chart. They are also used to define the edges of a knit panel, and can optionally have shaping instructions attached to them (right).*

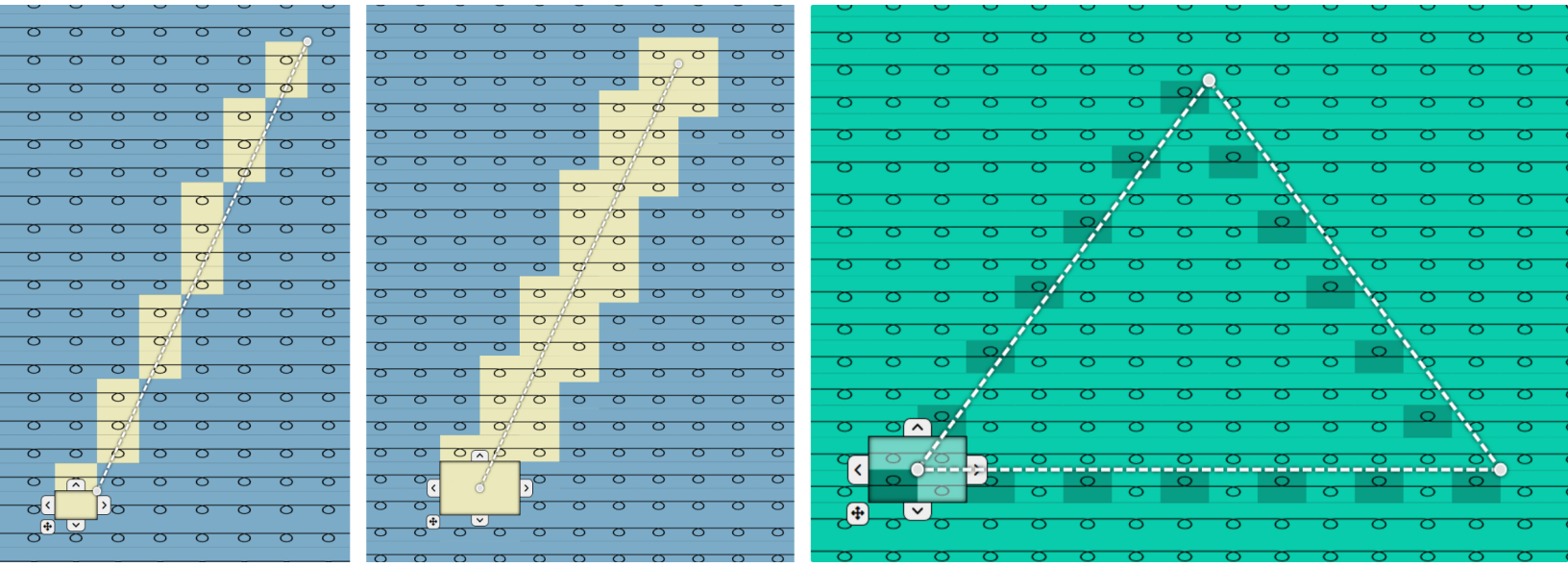


Figure 5.6 *Paths can be used to position stitch and yarn blocks. We can change path appearance by changing the size of the associated block, e.g., to make a single-stitch wide yarn path (top left) into a two-stitch wide yarn path. The same applies for stitch paths (bottom), where we can create a dotted stitch path by adding extra transparent space at the edge of the stitch block.*

Design Interface

Our design tool (Figure 5.7) consists of a two-view interface showing our compact chart editor on the left and a yarn-level visualization on the right.

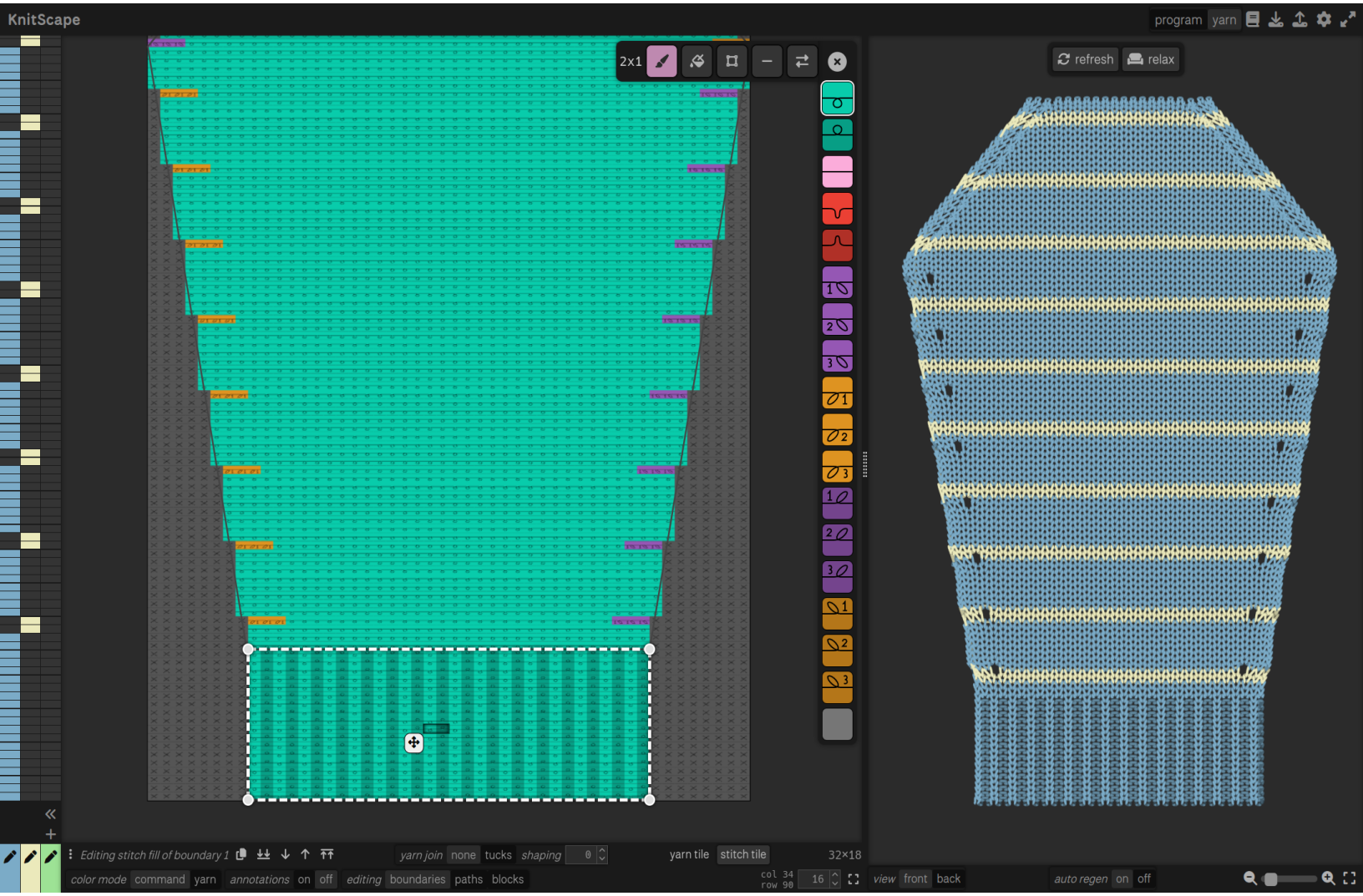


Figure 5.7 *Our design interface consists of a split-pane view of a chart editor and yarn simulation.*

Charting Pane

The charting pane contains a layered view of our overlaid stitch and yarn charts and vector primitives, as well as other interface elements and toolbars which display contextual information. Users manipulate vector elements in the charting pane by dragging on points and segments of vector elements (Figures 5.5 and 5.6) and editing their associated blocks (Figure 5.4, top). Whenever a boundary, path, or block is changed, we rasterize pattern elements to create the underlying stitch and yarn charts (our rasterization process is described further in Section 5). This means that the user never makes direct, destructive edits to a global instruction chart. This is a core advantage to our blended primitives approach: changes to a chart which would be labor-intensive under a traditional pixel-paint workflow (e.g., resizing overall dimensions or translating groups of operations) are instead rapid, incremental, and quickly reversible, recalling foundational principles in the design of interfaces for direct manipulation [218].

The appearance of the chart depends on the active color mode. Each chart cell always contains the symbol of its assigned operation from the stitch chart, but symbols alone can be hard to differentiate (especially when zoomed out), so we offer two color modes for chart editing. In *command mode* each cell is colored according to operations assigned in the stitch chart (shown in Figure 5.3), and in *yarn mode* each cell is colored according to yarns assigned in the yarn chart. The user can quickly switch between these modes using hotkeys or a button in the bottom toolbar. When the yarn color mode is active, we shade back-bed operations to give them the appearance of depth, which can help give the impression of a rib (an example of this can be seen in Figure 5.13, B). To enable some visual feedback of yarn color when the command color mode is active, the yarn sequence pane at the far left always shows which yarns are assigned to each row of the chart. The user can also recolor the current yarn palette in this pane, which updates both the chart and yarn visualization. Finally, the bottom toolbar enables the user to switch between three editing modes that control which elements are able to be selected and edited: boundary, path, and block.

Yarn visualization pane

Our interface includes a yarn-level visualization to provide constant feedback on design decisions. The yarn visualization consists of layered yarn segment paths, shaded

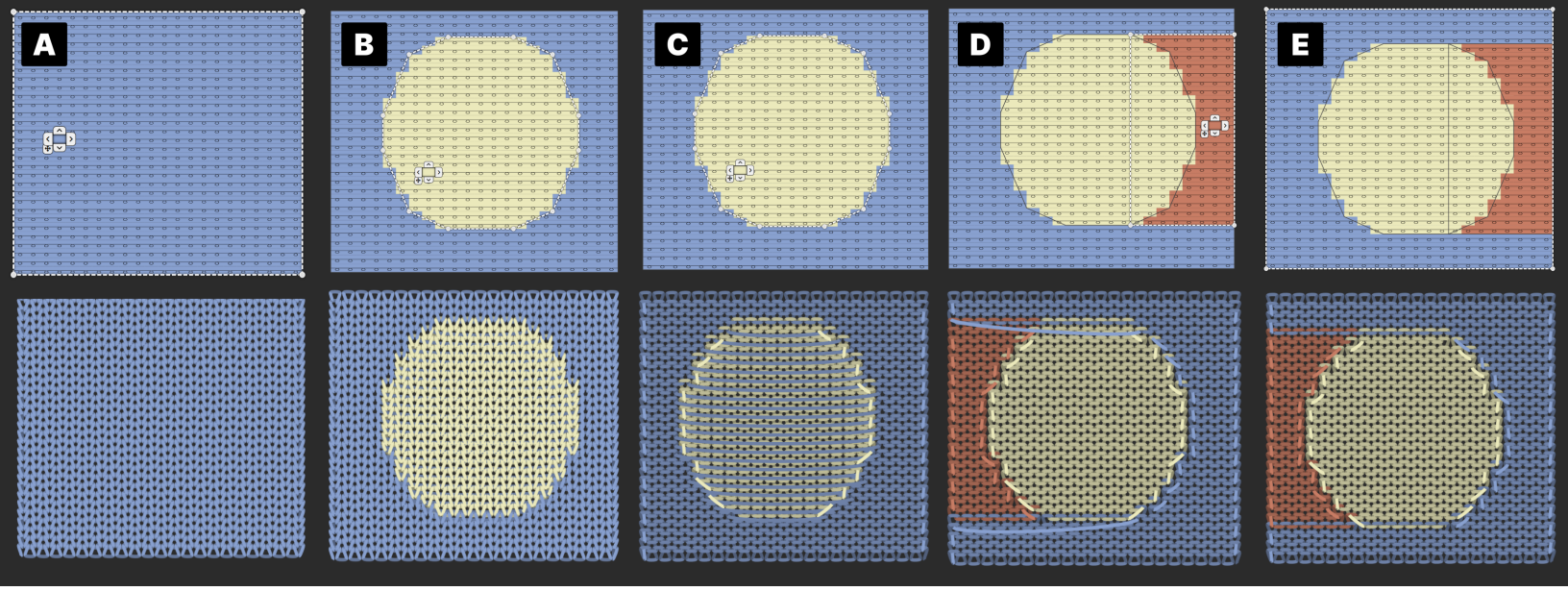


Figure 5.8 *Intarsia knitting: The yarn visualization provides feedback on design decisions made in the charting environment. In this example, it shows the position of yarn floats across the back of the fabric while designing an intarsia pattern.*

to give the illusion of depth. It can be viewed from the technical front or back, which can be helpful when inspecting float position on the back of the fabric. The visualization updates, (regenerating yarn topology and redrawing yarn segments), on edits in the charting interface. For smaller patterns (e.g., below 100x100 stitches) these updates are almost instantaneous, but topology generation slows down for larger patterns and those which include multiple yarns and/or short row shaping (and depends on device capability). Therefore we also support a visualization mode where the topology is regenerated on user request, which can be helpful when designing larger or more complex patterns which include multiple yarns. The yarn visualization includes a 2D relaxation simulation where contacts between yarns are modeled as a particle-spring system. The relaxation can be run on request via a button in the visualization toolbar, providing a preview of how yarn segments will deform once knit.

Example Workflow

In this section we describe an example workflow in which we design a simple intarsia pattern, demonstrating how our yarn visualization provides rapid feedback on edits made in our charting interface (Figure 5.8). On creating a new pattern, the workspace opens in boundary edit mode to show a stockinette swatch: a rectangular boundary with a knit stitch fill and a single yarn fill (Figure 5.8, A). To begin an intarsia design, the user places control points to define a new boundary (in this case, a circle). By

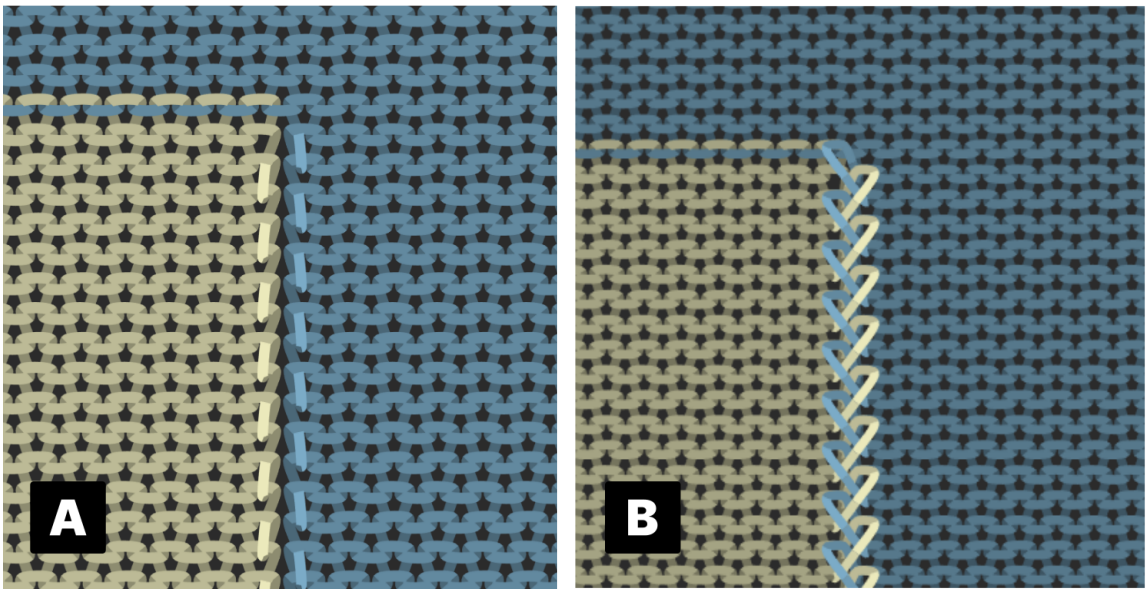


Figure 5.9 *Boundary conditions: The "join" setting controls how yarns are joined at a boundary. When "none" is selected, no change is made. In the case of a vertical join, this will produce a slit (A). The "tucks" option adds a tuck to the previous color in the row, depending on the direction the carriage is moving. These are visible in the simulation view by viewing the back (B).*

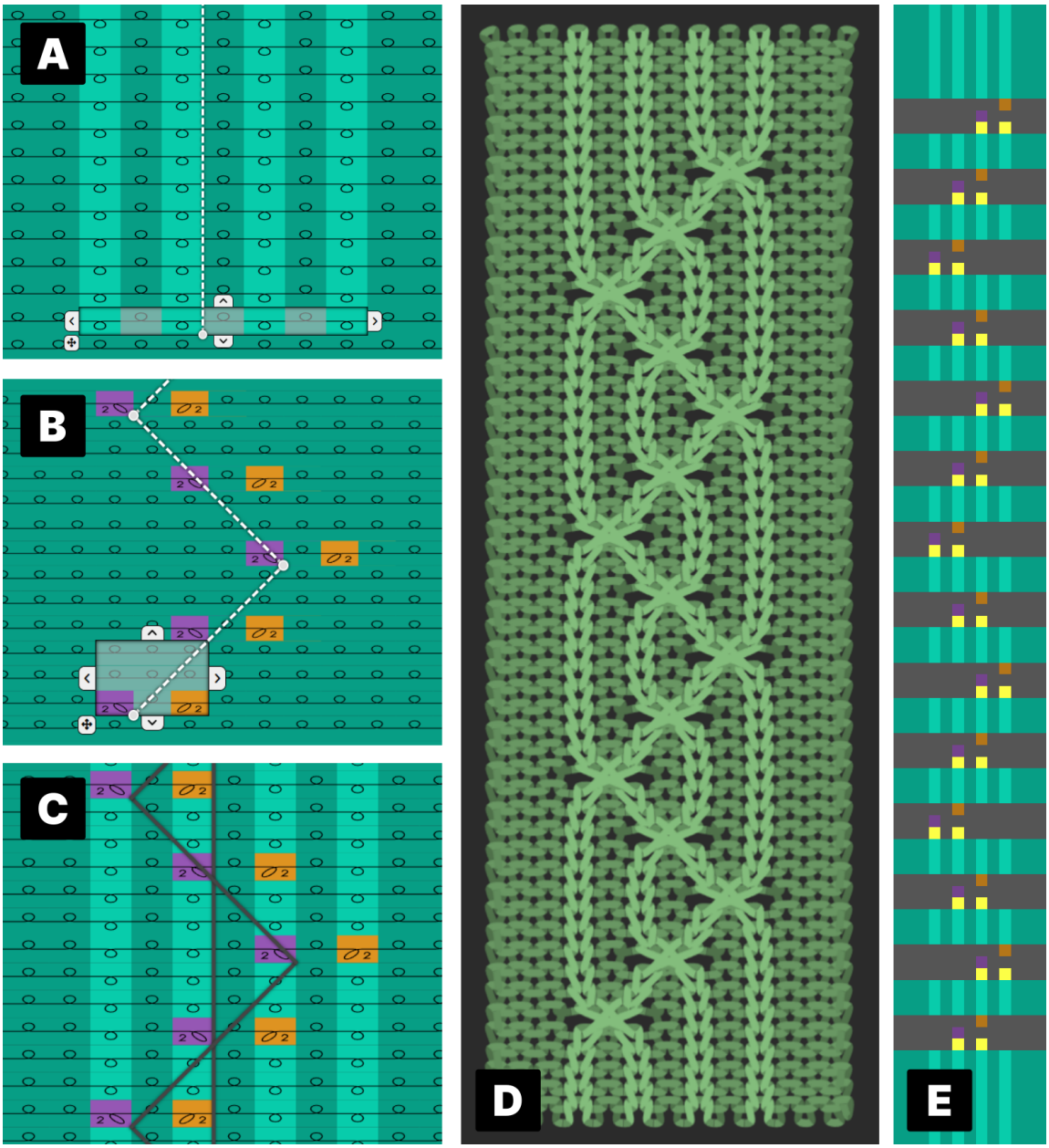


Figure 5.10 *Cable knitting: We designed this cabled lattice by first using a stitch path to make a 1x1 rib (A). A second stitch path is used to place transfers in a zig-zag (B), which we overlay on the rib definition (C). Together, these create a lattice effect which can be seen in the yarn visualization (D) and exported for knitting (E).*

default, all boundaries defined after the first boundary have a transparent stitch and yarn fill, meaning that they will inherit the operations and yarn assignments from prior layers. Therefore, to make the circle a contrasting color, the user assigns a contrasting yarn fill (Figure 5.8, B). These modifications immediately update the yarn visualization, which now shows the light yellow circle. However, "flipping" the swatch in the visualization pane shows that the blue yarn leaves long floats as it crosses behind the yellow circle (Figure 5.8, C).

While designing intarsia, these floats can be removed by using another yarn to knit the blue segments which appear to the right of the circle, preventing the yarn from needing to float across the yellow circle. In our editor, this can be accomplished by simply drawing a new boundary that is the same height as the circle and moving it to the layer below the circle (Figure 5.8, D, left). This removes most of the long blue floats, but two remain as the blue yarn ends up on the wrong side of the pattern due to an odd number of rows. In this example, this is addressed by shifting the circle and the red boundary down by one row, removing the floats (Figure 5.8, E). If this pattern was knit as-is, there would be gaps where the yarns meet at each row. This is addressed by enabling the "tucks" option on the yarn boundaries to add tucks when two yarns meet at a boundary. Details on this are shown in Figure 5.9.

Demonstration Examples

In this section we demonstrate how our blended primitives approach can be used to recreate common knitting elements.

Cables, Braids, and Twists

Cable, braid, and twist textures are produced by selectively crossing groups of stitches to give the appearance of multiple interwoven strands of fabric. Crossed stitches on their own do not have much depth, and cable strands are typically given more definition by using knit stitches for the main cable body and purl stitches as a background. There are multiple ways to design cable structures using blended primitives. We found

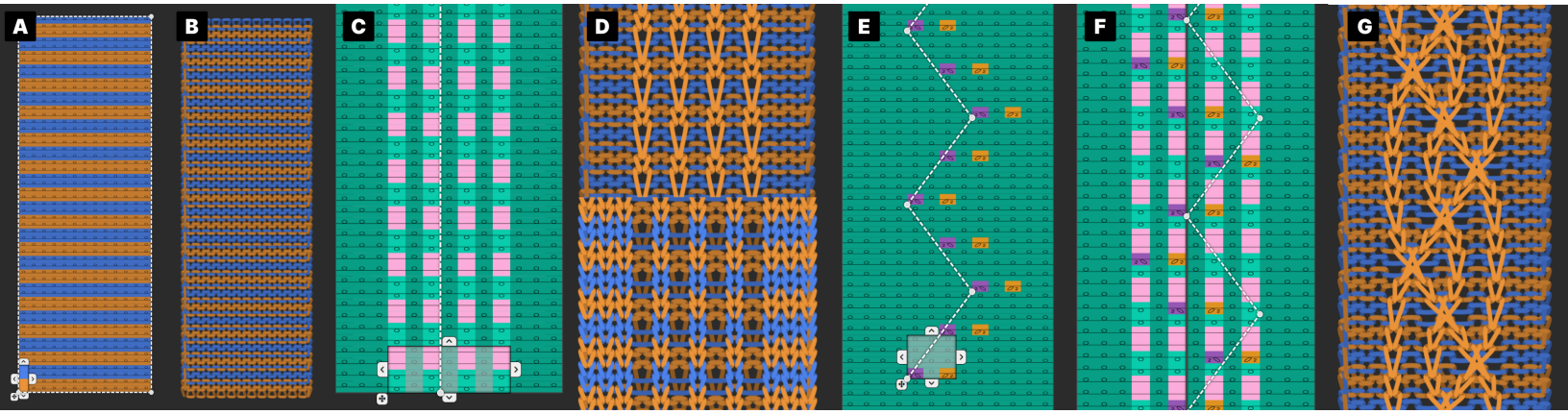


Figure 5.11 *Colorwork cable knitting: To design a colorwork cable lattice, we begin by (A) applying a striped yarn fill and purl stitch fill to a rectangular boundary. A (B) yarn-level visualization of the resulting knit gives constant feedback on design decisions. Switching to the operation view mode, we add a (C) stitch path to overlay a 1x1 rib texture which includes miss stitches every two rows. The (D, top) resulting rib only includes orange yarns, as the blue yarns are (D, bottom) floated across the back. Next, we create (E) a stitch path which tiles transfers in a zigzag pattern. When (F) overlaid on the rib texture, the (G) transfers move loops between the vertical lattice ribs to give them an interwoven effect.*

it easiest to control the base knit/purl texture separately from the transfers. We did this by defining separate stitch paths for the knit border and the transfers. Example cable, braid, and twists can be seen in Figures 5.1 and 5.10. By overlaying the texture and transfer paths atop a background with a striped yarn fill, we can also design and simulate colorwork cable structures such as the orange and blue cable lattice in Figure 5.11.

Fair Isle Colorwork

Fair Isle colorwork (also known as stranded colorwork or float jacquard) is a technique for producing color patterns where multiple yarns are used across a whole row of knitting. For each row, the yarn which should appear on the front of the fabric knits, and the non-visible yarn (or yarns) floats behind the knit stitches. Figure 5.13 shows an example where multiple colorwork motifs are used to pattern a “doodle cowl”, a type of pattern which is currently popular in the hand knitting community as a sampler of themed colorwork motifs. Our yarn separation can theoretically handle an arbitrary

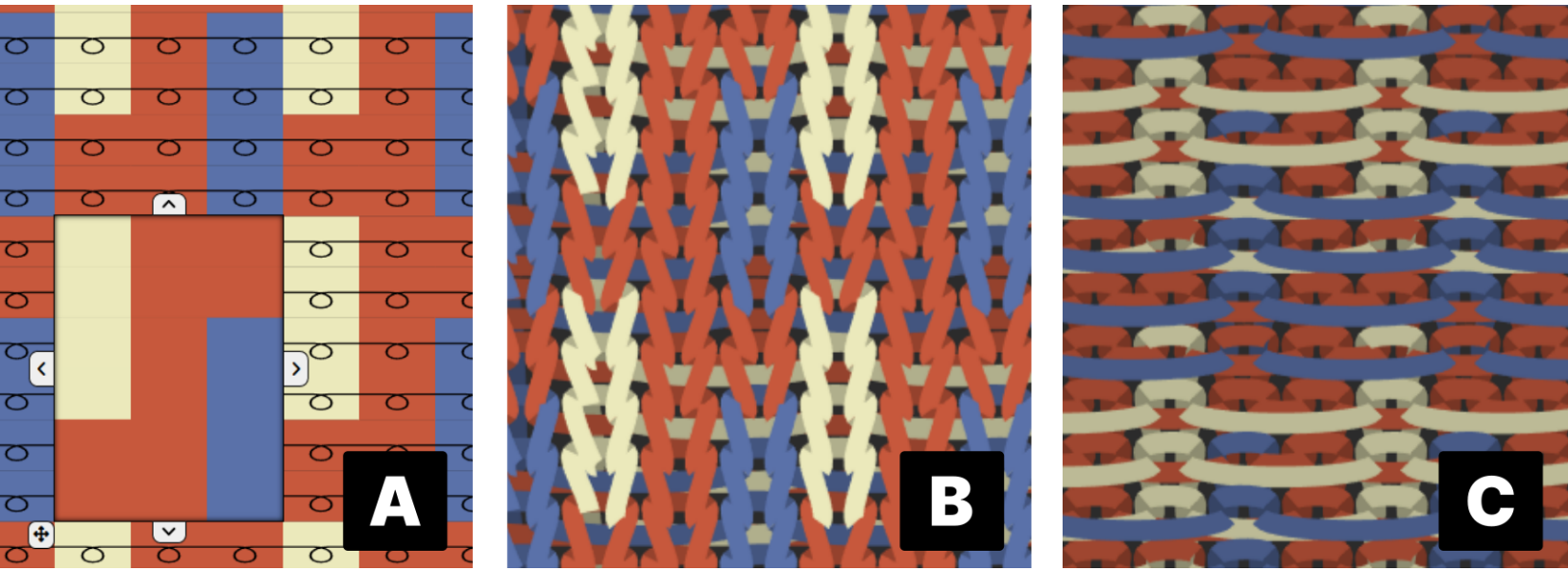


Figure 5.12 *Colorwork: We support colorwork patterns which include more than two colors per row, such as this motif which includes three colors per row (A). The fabric is shown in the yarn visualization (B), and the resulting floats can be inspected by viewing the back (C).*

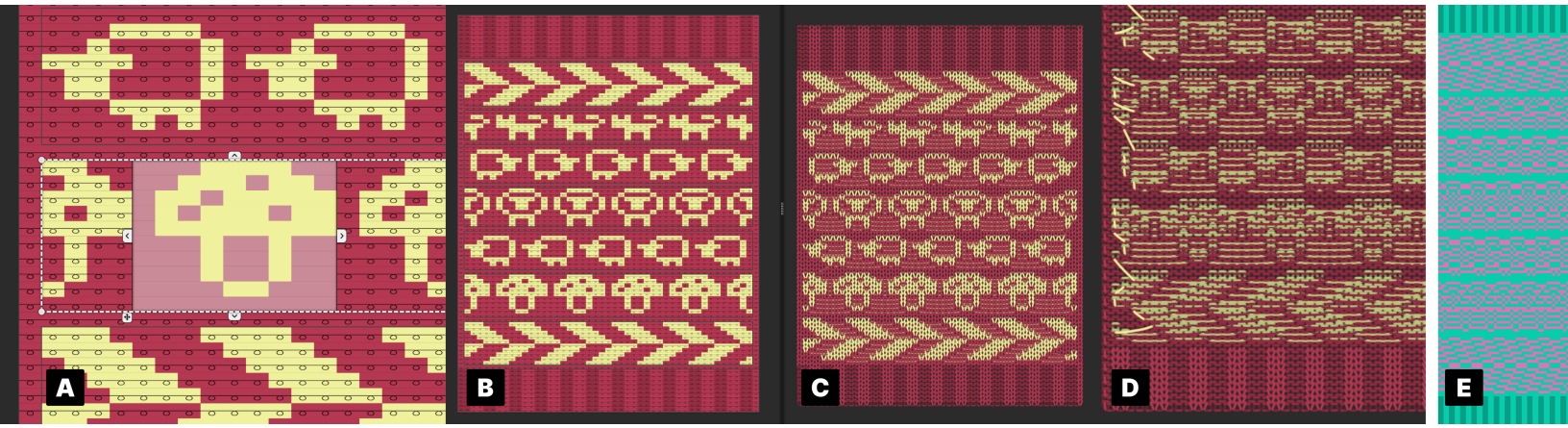


Figure 5.13 *Colorwork repeats: Yarn blocks are tiled to fill their boundaries (A). Boundaries enable easy positioning of colorwork and texture motifs, such as to design this sheep-inspired doodle cowl with ribbed edge (B). The simulation pane shows a yarn-level visualization of the resulting knit (C). Yarn floats can be viewed by flipping the visualization to view the technical back (D). The resulting time-needle view can be seen in (E).*

number of yarns per row, although there are practical limitations to this depending on maximum float length. Figure 5.12 shows a colorwork pattern which includes three colors per row. The back of the fabric can be examined in the yarn simulation, which enables inspection of the resulting floats.

Intarsia Colorwork

Intarsia is a colorwork technique appropriate for patterns which include large, unbroken blocks of color. A separate yarn feed is used to knit each segment of color which appears in a row. This means that a circle which appears in the center of a fabric requires three separate yarns (Figure 5.8), and the "U" in UIST requires five separate yarns (Figure 5.14). Our editor supports design of intarsia patterns via yarn paths and yarn fills. To prevent gaps from forming between adjacent blocks of color, the yarns must be joined at the color changes. One strategy for joining is to add tucks to the preceding color segment. In our editor, join conditions are specified by selecting the "tucks" join mode option on a boundary or path (Figure 5.9). By placing a tuck on either the front or back bed depending on the position of the joined stitch, we also support design of textured intarsia patterns.

Lace

Lace is knit by using different operations such as transfers to create delicate patterns of holes. Blended primitives offer multiple approaches to the design of lace patterns, which can be specified using a traditional block-based repeat as a stitch fill of a boundary-defined region (Figure 5.15, left) or by using a stitch path to tile a block of instructions along a vector-based path (Figure 5.15, right). The stitch fill approach may be more appropriate for all-over textures, while stitch paths are better suited to the design of individual motifs.

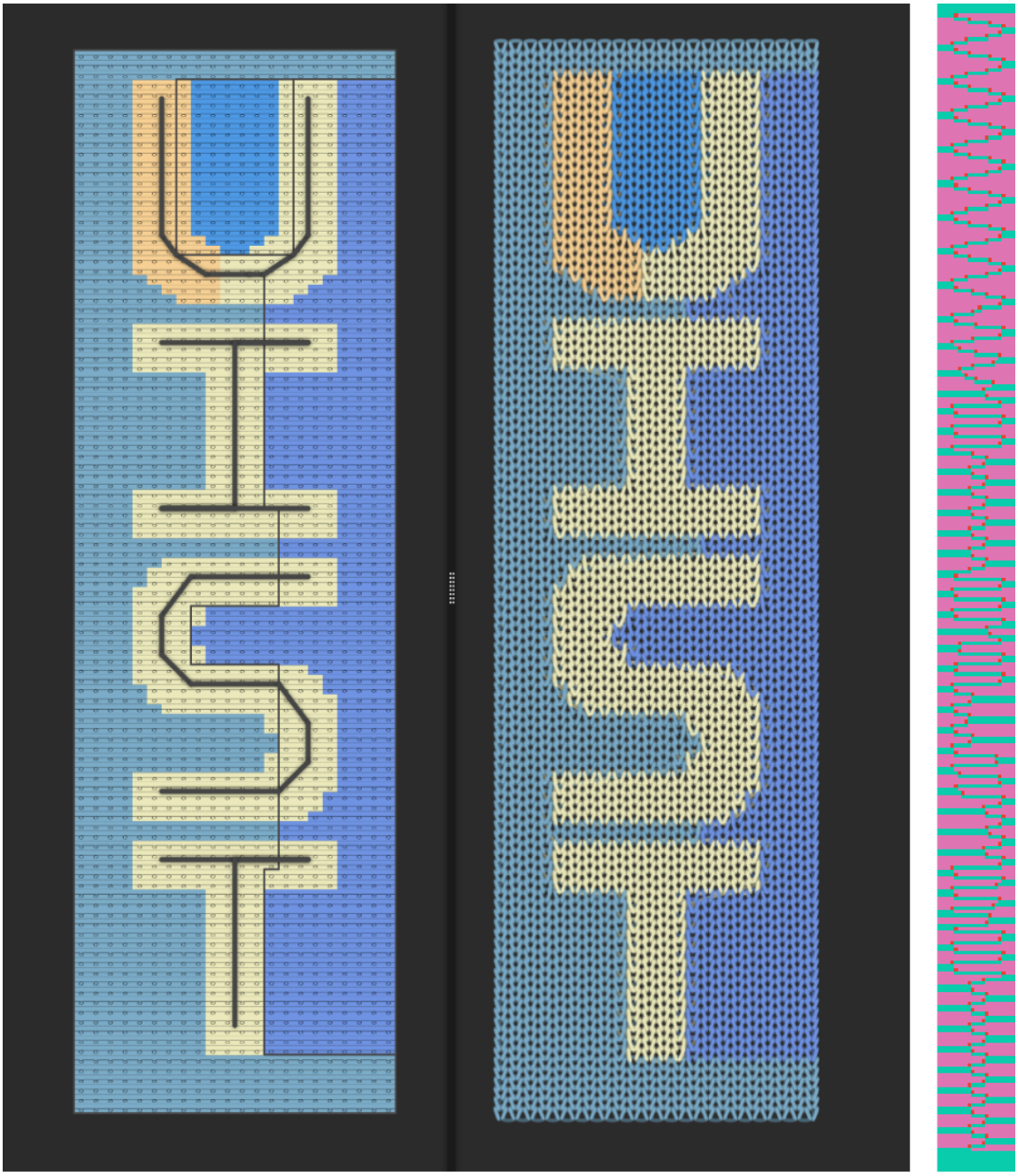


Figure 5.14 *Path Intarsia: Whereas Figure 5.8 shows a patch defined with a circle boundary, this example uses a path with a corresponding stitch block to create the letters UIST. "U" in "UIST" requires five separate yarns to knit it with the intarsia technique: three in the background color (one for each side plus one in the middle) and two for the letter color (one for each leg).*

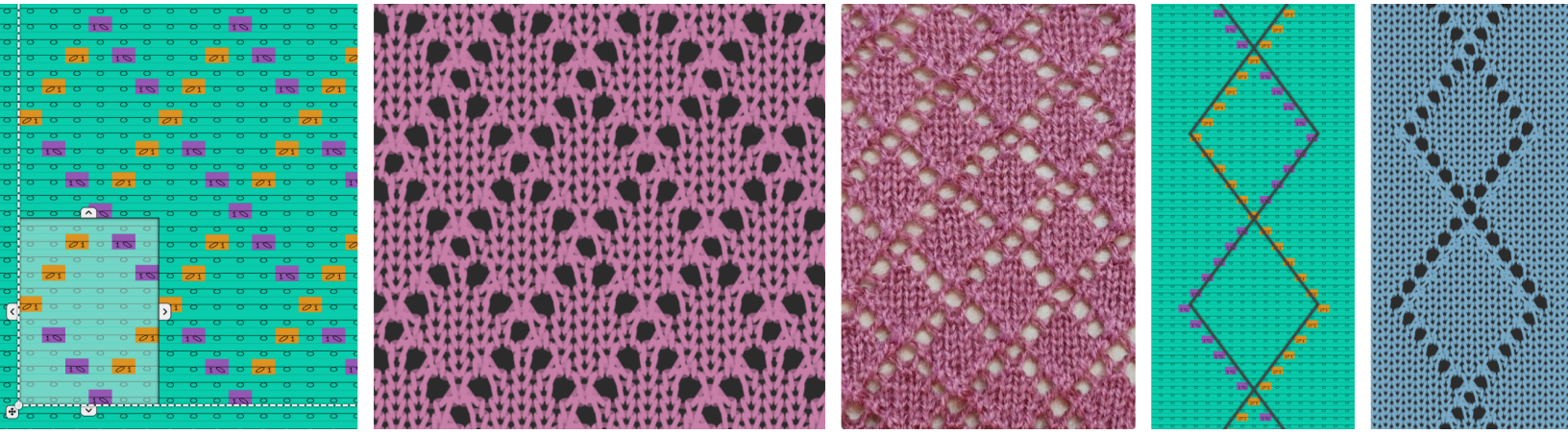


Figure 5.15 *Lace knitting: We support multiple approaches to lace design. Lace patterns can be designed by using a stitch block to fill a boundary (left). This allows the quick application of a repeating lace texture to a region. Stitch paths can also be used to design lace textures (right). This method is well-suited to individual lace motifs.*

Further Applications: Sensors and Folds

Blended primitives enable us to build abstractions for applications beyond the more traditional knitting techniques described in the previous section. We describe two example applications in this section: intarsia-based sensing structures and origami-inspired pleat patterns.

Origami-Inspired Pleat Patterns

One technique to design knitted folds or pleats is to take advantage of the curling behavior inherent to knit fabric: surfaces made entirely of knit stitches have a natural tendency to curl towards the back. Staggering knit and purl stitches can even out this curling behavior and make a fabric lie flat. Therefore, to design a pleat, we can use a base pattern with minimal curl and strategically place knit or purl stitches where we want the fabric to curl. Blended primitives enables us to abstract this core idea into reusable abstractions for mountain and valley folds, which we have used to design origami pleat patterns (Figure 5.16). Our origami patterns consist of a background boundary with a garter stitch fill (alternating rows of knit and purl stitches). We then

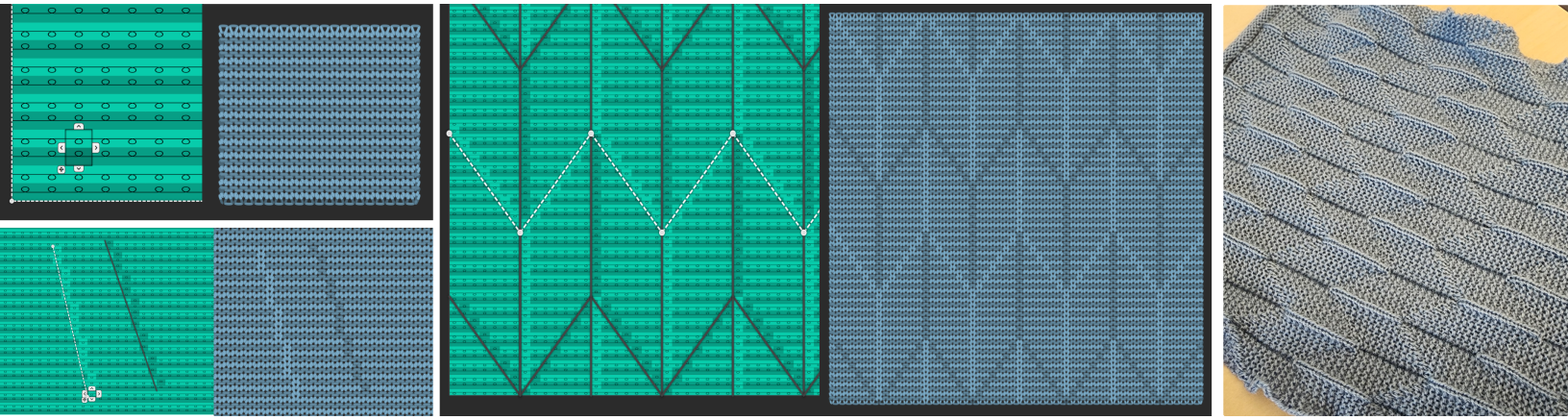


Figure 5.16 *Self-folding knit pleats: Origami pleat patterns can be defined by layering knit and purl stitch paths over a garter stitch region. This takes advantage of different stitchs' tendency to deform to create self-folding patterns.*

define a mountain fold as a knit stitch path, and a valley fold as a purl stitch path. Using these stitch paths, we can draw mountain and valley folds atop the garter background. This shows how our tool supports the design of folding and shape-changing knits.

Intarsia-based Sensing Structures

Knit textiles can be used to sense environmental changes by incorporating conductive and other active yarns into the fabric structure. These sensing structures can be used in a variety of ways to sense stretch, pressure, touch, humidity, and more. This has been an area of increasing interest to researchers exploring the intersection of HCI and wearable interfaces [192, 147, 154, 248, 7, 180]. However, the challenges of low-level knit programming extends to this emerging domain. Our design tool is not specific to the design of knit sensors, but our yarn paths and fills support the design of single-layer multi-yarn sensing structures based on the intarsia technique.

We demonstrate how to knit two types of intarsia-based sensors: capacitive touch sensors, which we use to create the volume control pad in Figure 5.18, and interdigital electrode sensors, which we use to sense pressure (to detect foot posture with a pressure-sensing sock in Figure 5.17) and stretch/bend (used in the bend-sensing mitten in Figure 5.19). For the latter sensor type, interdigital electrodes are positioned

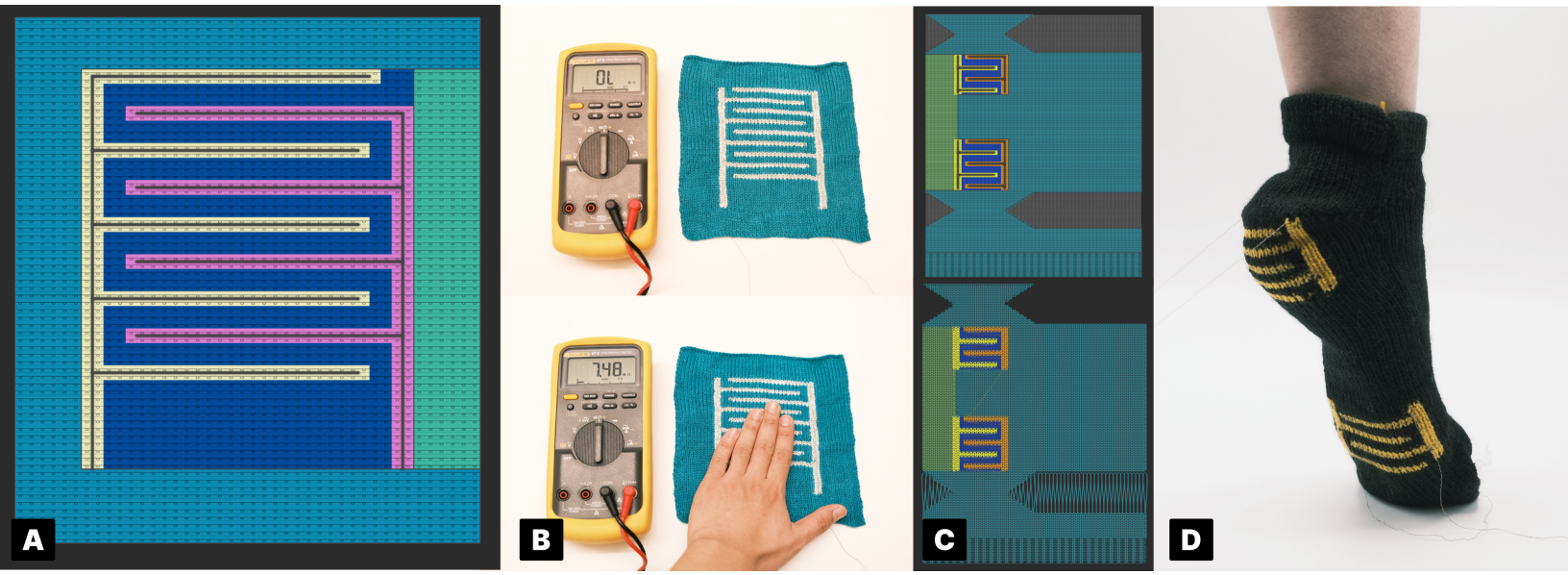


Figure 5.17 *Knit pressure sensors: Yarn paths and fills can be used to design an interdigital electrode structure in our editor, which can be knit via the intarsia technique (A). Note that we have used contrasting colors for the five separate yarns for visualization purposes, but all yarns could be the same color when knit to hide the presence of the sensor. The interdigital electrode structure can be used to sense pressure when in direct contact with skin (B), and positioned on a sock pattern in our editing environment (C, top) and visualized (C, bottom). Stretched loops in the yarn visualization indicate loop connectivity across short rows required to shape the sock heel. The knit sock (D) can sense changes in pressure at the heel and toe.*

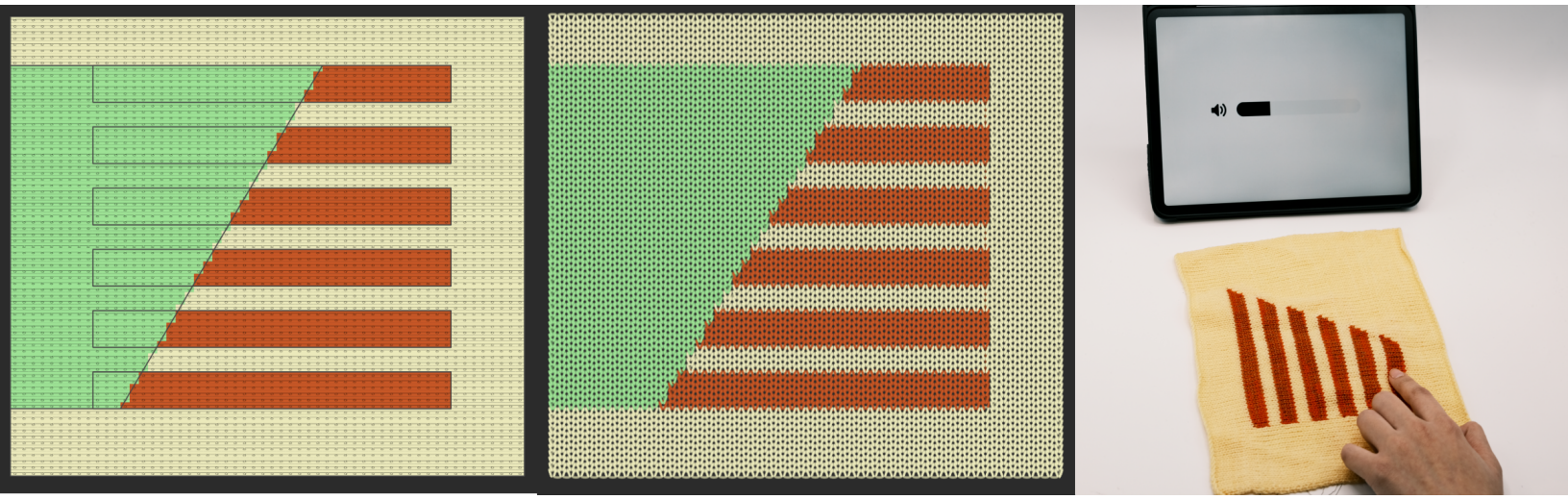


Figure 5.18 *Knit sensors: Knitting an intarsia pattern with conductive yarns creates distinct conductive regions that can be used for capacitive touch sensing.*



Figure 5.19 *Knit bend sensors: The interdigital electrode structure used for the sock shown in Figure 5.17 can also be used to sense stretch, used here as part of a bend-sensing mitten.*

against the skin, where higher pressure, resulting either from direct contact (sock) or from localized bending (mitten), will augment skin-electrode contact and thus reduce the resistance measured across the electrodes. We knit these samples by running a conductive embroidery thread (Madeira HC-40) in the same yarn feed as the yarns which were intended to be conductive. This shows how our design tool supports the integration of yarns into localized sensor regions.

System Implementation

The blended primitives design tool is browser-based and written in client-side JavaScript. For a link to the open-source code repository and hosted version of the tool, please visit the project page[1]. The tool consists of a split-pane view of our novel pattern-editing interface and a 2D yarn-level simulation based on the open-source KnitScape system [246]. In this section we describe our implementation of a novel pattern editing interface for design with blended raster/vector primitives as well as our extensions to the KnitScape yarn model and simulation. We tested our export format on the Kniterate machine, which supports import of an instruction chart and associated yarn sequence.

Stitch and Yarn Chart Rasterization

The pattern data structure consists of three arrays (for boundaries, paths, and free blocks). We begin by computing the bounding box for all boundaries, and use its dimensions it to initialize two bitmap charts which will hold the operations and yarn assignments for each cell. We refer to these as the *stitch chart* (SC) and the *yarn chart* (YC), respectively. The stitch chart is initially filled with the empty stitch, and the yarn chart is filled with the empty yarn. We then rasterize the elements of the boundary, path, and free block arrays to the stitch and yarn charts. The rasterized charts are overlayed in our chart editor to determine which symbols and colors appear in each cell.

One important decision we made was that boundary and path coordinates refer

[1] https://depts.washington.edu/machines/projects/blended-primitives/

to the lower left corner of a cell rather than its center point. This is to ensure that paths and boundaries will maintain their geometric magnitude when rasterized, i.e., a line from $[0, 0]$ to $[20, 0]$ will rasterize to be 20 cells long (as opposed to 21 using center points). In our implementation, only the interior of the path is rasterized, meaning that the cell at $[20, 0]$ will not be filled. We felt this was better suited to chart editing as it is very common to specify dimensions in stitches (width) and rows (height). Additionally, defining coordinates in this way makes a clear distinction for stitch assignment when adjacent regions share a boundary edge.

To generate the machine instructions and yarn topology for the yarn simulation, each yarn must be scheduled in its own carriage pass. This requires further processing of the stitch and yarn charts and is handled during our yarn separation and transfer planning step described in subsection 5.

Boundary Rasterization

We rasterize each polyline boundary using a scanline fill approach modified to incorporate some knitting-specific considerations. Pseudocode can be found in Appendix C. At a high level we follow a standard scanline polygon fill algorithm [82], initializing an edge table (ET) and active edge table (AET) and processing the intersections between a horizontal scanline and any active edges in pairs. In order to better approximate the boundary, we use a scanline located halfway up the cell ($y + 0.5$). We do this by offsetting the edge's current x value during edge table creation, setting it to $x1 + (dx/2)$. For boundaries that have shaping instructions, each edge also stores the last x value rounded to the nearest integer ($xLast$), which we examine to detect when rows are increasing or decreasing.

During the main loop, we round the scanline intersection of each edge pair to the nearest integer to find the start and end points for the pattern fill ($x1$ and $x2$). We then fill the space between $x1$ and $x2$ in the stitch and yarn charts by copying values from the respective blocks, skipping locations where the block includes the "transparent" stitch or yarn. The interior space is patterned using a simple 2d repeat of the base block. It would be straightforward to extend support to other repeat modes (e.g., ones that include block reflection or an offset between pattern rows or columns). If the boundary has shaping instructions, we examine the relationship between $x1$ and $x2$ and the $lastX$ value of their respective edges and write fashioning instructions

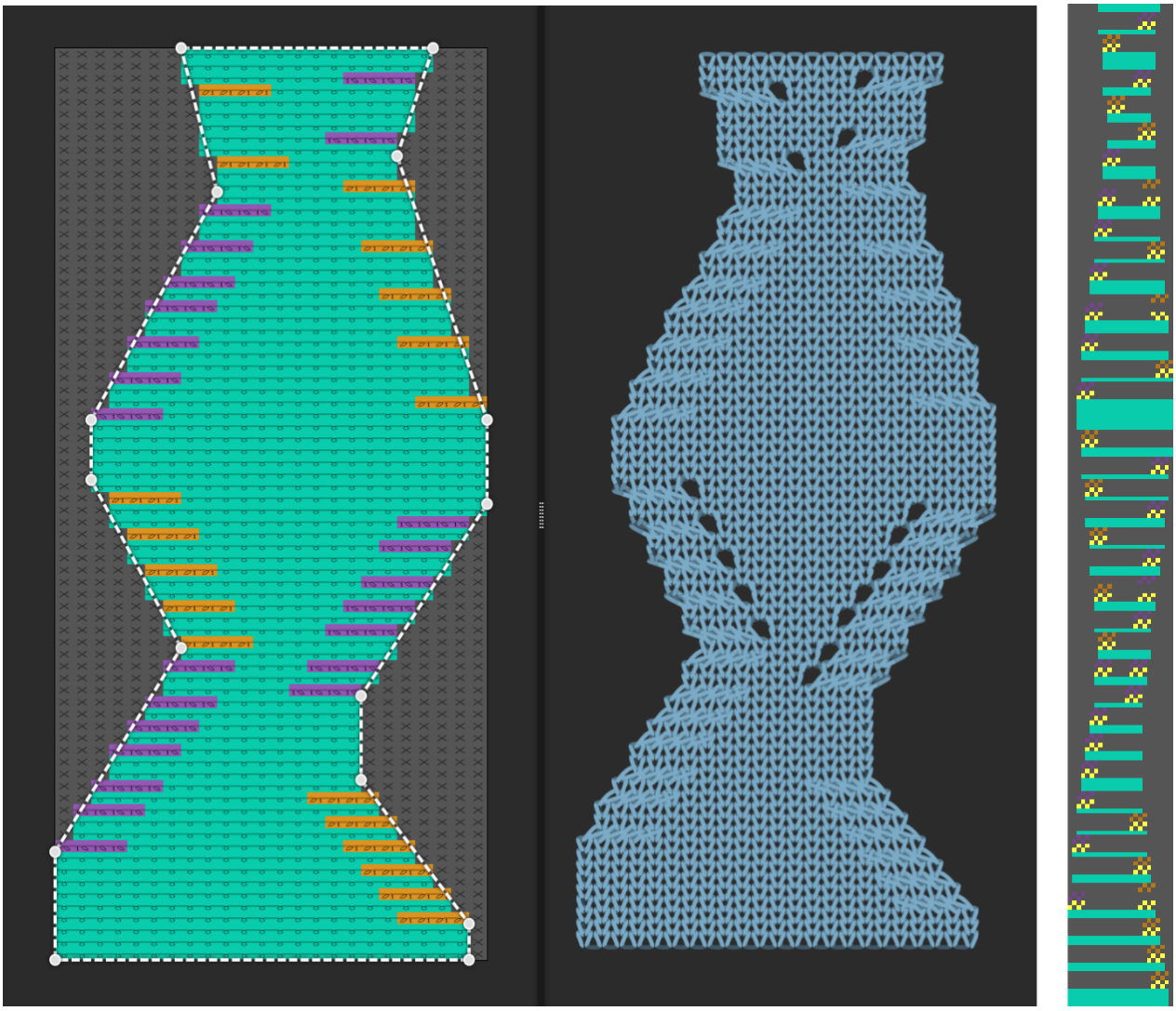


Figure 5.20 *When a boundary includes shaping, fashioning instructions are written where rows increase and decrease and update automatically as the boundary is reshaped. Pictured here is an oddly-shaped boundary in the charting interface (left), the resulting yarn simulation (center), and the time-needle view which includes scheduled transfers (right).*

(transfers) to the prior row to account for a difference: either moving a group of loops out (an increase) or in (a decrease). Shaping instructions are only applied to single-stitch increases and decreases, i.e., edges where $-1 \leq 1/m \leq 1$.

Path Rasterization

We rasterize each polyline path using a Bresenham approach [24], again modified to incorporate some knitting-specific considerations. Pseudocode can be found in Appendix C. We traverse each polyline segment from [$x0$, $y0$] to [$x1$, $y1$] and draw the associated stitch and yarn blocks depending on the tile mode of the path.

The "tiled" mode examines the last position of the block and only plots it if the difference in x or y is greater than the block width or height, respectively (an example of this can be seen in Figure 5.10-B). The "dx" and "dy" modes specify whether the tile spacing should iterate in the horizontal or vertical direction. This can be helpful in cases where block iterations require exact spacing over a number of rows (or columns), but the other dimension does not matter. Finally, the "overlap" mode plots the whole block at every location.

Because path coordinates (like boundaries) refer to the lower-left corner of a cell rather than the center point, the vector path is not perfectly centered for paths using a block of an odd cell width. While this may be off-putting to the perfectionists among us, it enables fine control of the path offset (e.g., to position all iterations of a block on the inside of a boundary.

Yarn Separation and Transfer Planning

When machine knitting complex patterns, loops often must be transferred from a front bed to a back bed and vice versa. We handle transfer planning similarly to the school-bus approach for the flat lace transfer problem described by Lin et al. [139]. Full implementation can be found in our source code repository. However, instead of transferring all loops between beds, we only transfer those with target offsets. A simple example demonstrating our transfer planner can be seen in Figure 5.21. First, we transfer all loops with a target offset to the back bed (bright yellow), staggering them so as to not transfer adjacent loops in the same pass of the carriage. Then, for each

target offset (in this case -1 and 1) we transfer the loops back to the main bed, again staggering them so as to not transfer adjacent loops.

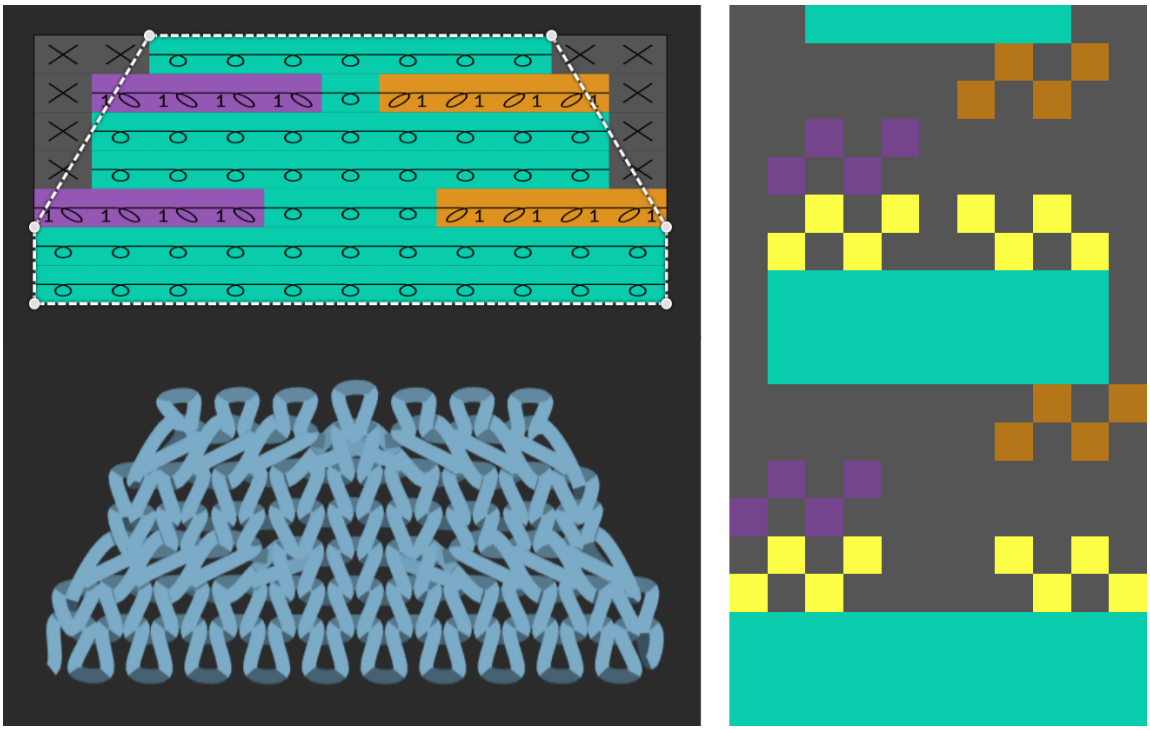


Figure 5.21 *We stagger loop transfers so as to not transfer adjacent loops in the same pass of the carriage.*

Yarn Topology and Simulation

Our yarn simulation extends the open-source KnitScape system [246], which was developed for the design and simulation of slip and tuck colorwork patterns. The yarn simulation uses an approach proposed by Kapllani et al. [109] to generate a graph of yarn topology from a chart of knitting instructions that encodes the order in which a yarn visits contact nodes where it is intertwined. To draw the yarns, leg and loop segments of a spline which is traced around the contact nodes are separated onto layers and shaded to give the illusion of depth. To preview the deformation caused by loops which have been stretched across multiple rows of knitting, the visualization also includes a particle-spring simulation where each yarn segment tries to contract to a rest

length based on the distance it extends from its original row of instructions. We made a number of extensions to the topology model and yarn simulation in order to support patterns which include transfers (e.g., cables, lace, and fully-fashioned shaping), and patterns which include multiple yarns per row (e.g., intarsia or fair isle colorwork).

Composite Transfers

Extending the existing knit, purl, slip, and tuck operation set, we implemented a composite "knit then transfer" operation where a loop is knit on the front or back bed and will be moved to its target location on the next transfer pass. Our full operation set can be seen in Figure 5.3. Currently, we have not implemented commands which enable control over the order in which loops end up in their target location (which can affect surface appearance in some cases), and this is determined by carriage pass direction. We think it would be straightforward to implement additional composite operations which enable finer control over transfer planning, and that this should be a priority for future work. In order to visualize the greater depth introduced by a transferred loop, we also extended the existing yarn visualization to support an arbitrary number of shaded layers determined by the maximum stack of yarns at any location in a pattern. To determine the correct stacking order for transferred loops, we implemented the approach described by Kapllani et al. [110]. Our full implementation is included in our open-source code repository.

Multiple Yarns

In order to support patterns which include multiple yarns per pattern row, we had to extend the existing topology model to account for multiple yarn paths and arbitrary carriage pass direction. The direction of the carriage determines the order in which nodes are processed during creation of the topology graph. When multiple yarns are assigned to a row (e.g., in intarsia patterns) the pass direction of a particular row will depend on which side the yarn in use was previously. This may result in arbitrary pass directions (e.g., right-right-right-left-left-right). We accounted for this by creating an array of carriage pass directions during yarn separation, and using that during the topology graph and yarn path creation to determine which direction the yarn would be moving in that row. To support multiple yarns in a pattern, we create a separate yarn path for each yarn and use the yarn sequence produced during yarn separation

to determine which yarn's path new nodes should be added to.

Challenges and Opportunities

Our demonstration examples show that a 2D chart-based notation such as ours can be powerful for designing mid-level texture and colorwork effects, but some design tasks (like 3D shaping) are challenging due to the convoluted relationship between a 2D notation and 3D result. While it is possible to use our notation to design surfaces with short row shaping (such as the sock in Figure 5.17), a 3D shaping primitive like a tube [152] or mesh [170] may be better suited to that task. However, changes to fabric structure can significantly impact the fabric gauge (e.g., some textures are short and wide while others are tall and narrow), which may need to be accounted for in the overall shape specification. This poses a practical challenge when building knit design interfaces: overall shape, colorwork/texture patterning, *and* the interaction between them determine the aesthetic and functional qualities of the final knit.

Future design tools must investigate how to support creative exploration of these different dimensions–and their interactions–in tandem. We don't think that future knitting design tools should prioritize reconciling disparate, task-specific representations under one notation system, as we think such a system would risk supporting some tasks far better than others (or worse: being "bad at everything"). We think future interfaces should instead focus on enabling someone to visualize and rapidly work *across* task-specific notations, such as by providing multiple synchronized editing views. As an example, we can imagine a future version of our system which includes a 3D stitch mesh editing pane, in which a stitch mesh [170] could be used to drive a base "shaping boundary" in our chart editor. Stitch and yarn paths could then be defined in the chart editor and visualized on the mesh surface to give a sense of their position in the global fabric. Another way to accomplish this in the context of our current interface would be to develop domain-specific block editors which can compile to our charting language. For example, a "brioche editor" which uses Albaugh et al. [9]'s brioche notation could be opened in place of our current stitch block editor.

While we have not formally tested the limits of our approach for fabric size, our tool is performant (i.e., changes to the chart update the visualization with minimal

delay [<50ms]) on our setup (Chromium on an i7 laptop with integrated graphics) for charts below 200x200 cells. Currently, update lag is largely due to a bottleneck when performing local searches to find the origin of unanchored yarn contacts during yarn topology generation, which particularly impacts intarsia patterns (a future iteration of our system may improve this by storing loop origin). However, we have tried to mitigate the impact of this delay on user experience by offloading computation to web workers where possible and adding a setting to toggle whether the visualization/yarn topology is updated on-the-fly (on input to the chart). We intend to continue to improve our simulation by incorporating advances made in recent work on efficient yarn-level simulation.

Discussion

Machine knitting is an incredibly powerful domain with a massive design space. In order to explore this design space, we need appropriate design representations that enable us to quickly iterate on various aspects of the knit object of interest. Our work focuses on supporting the design of new knit surface patterning effects such as cable structures, colorwork, and texture patterns. This is how our work differs from prior contributions which have largely focused on capturing overall fabric shape, such as tubes and sheets [152, 115, 112] and stitch meshes [169, 170]. We argue that design of fabric-level effects such as cables is a fundamentally different task than design of three-dimensional shaping, which has been the focus of prior work. We think that because such fabric structures deal closely with material behavior on the magnitude of an individual stitch, they require design representations which maintain easy access to low-level stitch operations while also supporting rapid manipulation of higher-level operation groups. We have implemented a design representation for knitting patterns we term *blended primitives*, which we contribute alongside our design tool.

Future directions for our work fall into three high-level categories, each of which is relevant to research in and beyond HCI. First, expanding the space of knit objects that can be specified and visualized in our system will be important to enable design of complex fabric structures, supporting ongoing development of novel functional fabrics. Currently, only patterned sheets can be designed in our editor. Through ex-

amples, we have demonstrated that a variety of single-layer sensing structures can be designed this way via intarsia patterning. However, due to a limitation of our underlying topology model we do not support double-sided or circular-knit fabrics, techniques which have been previously demonstrated to also be effective methods for fabricating knit sensors and actuators [147, 192, 191, 7, 148, 118]. We think it will be feasible to extend our editor to these areas with reasonable effort, and this is a priority for future work.

Second, providing designers with many different ways to specify and manipulate primitive elements will be crucial for those exploring the intersection of machine knitting and other domains. We were interested in building a (somewhat) vector-based design environment for knitting in part due to the many possible avenues for interoperability with existing tools and file formats. For example, we think it would be straightforward to implement support for a crease pattern format (such as [44]) that automatically defines stitch paths along fold creases, enabling knit origami patterns (such as the one in Figure 5.16) to be specified in origami-specific design tools and imported into our system. We anticipate that keeping an eye towards interoperability in future tool development will be helpful to those venturing into domains with few established design tools, such as shape-changing fibers [61].

Third, informing a designer of fabrication constraints and errors during design will always be an important aspect of a design tool. We think that the yarn-level visualization included in our tool is effective at helping designers navigate some of the challenges of low-level knit programming, such removing long floats in the example intarsia workflow shown in Figure 5.8. We think that the goals of a simulation used alongside a charting environment are different from those focused solely on rendering, and should be centered on providing effective feedback to the designers. For example, rendering fiber-level detail for more “realistic” yarns might distract from features of yarn topology which may be more relevant to designers, such as loop connectivity and float position.

Finally, while we have focused on design for machine knitting in this chapter, we want to recognize its deep roots in the craft in hand knitting, which continues to be beloved by millions around the globe. Hand knitting-oriented design and simulation tools would have the potential to reach and delight a massive audience. Our charting

language was designed with machine knitting in mind, but it already is compatible with a significant subset of hand-knit charting notation, which has had hundreds of years to mature and continues to be the primary form of visual pattern sharing among hand knitters [25]. We are very interested in building a dedicated hand-knitting version of our system, as we think that this would not only be a good way for hand knitters to take advantage of our blended primitives editing tools and synchronized yarn visualization, but it could also provide an alternative way to edit machine knitting patterns that would be accessible to those who are already fluent in chart notation.

Some related work has already begun in this space, e.g. by formalizing knitting shorthand [87]. The possibility space of hand knitting is far greater than that of machine knitting, so development of design tools that include some form of simulation presents some practical development hurdles and open research problems, although some mesh-based approaches exist [93]. However, such tools would also provide us with an easy way to import, visualize, and archive historic charted patterns. This could provide an important cultural snapshot of a craft which, despite its undeniable ubiquity, is often overlooked [150, 242].

Conclusion

We presented *blended primitives*, a design representation for knit fabric structures where a chart of knitting instructions and yarn assignments is rasterized from layers of vector boundaries and paths with associated raster stitch and yarn blocks. We further contribute a chart-based design tool which enables design and simulation of knit objects via blended primitives. We have demonstrated how blended primitives can be used to design a variety of common knit fabric structures, including fair isle and intarsia colorwork, lace, and cables. We have also demonstrated how our approach may be useful to support future work on the development of functional fabrics through example demonstrations of intarsia-based sensing structures and origami-inspired pleat patterns. We argue that the design of novel fabric-level structures, such as colorwork and texture patterns, requires different design representations than the 3d shaping-oriented primitives which have been the focus of prior work. We are excited to continue to improve our blended primitives editing environment and hope our open-source

contributions can enable future collaborations with researchers across domains.

Chapter 6

Swatchy Tools

Digital/physical creativity spans two worlds: the abstract digital world where designs are specified, and the concrete physical world where they play out. Working across the digital/physical divide involves grappling with *emergent behavior*: the surprising and often unintuitive effects which arise from low-level interactions between designs, precision, and materials.

Dominant trends in digital fabrication minimize emergence by abstracting the design of a thing away from the process through which it is made. However, as we saw on #PlotterTwitter, practitioners also value tools which enable them to explore emergence at the digital/physical divide. While tools for creativity have traditionally excelled at helping us explore possible designs, tools for digital/physical creativity should also help us explore what happens when those designs meet the physical world.

The previous chapters introduced tools for digital/physical creativity I have built, each of which foregrounds engagement with emergent behavior at the digital/physical divide via a different mode of exploration:

- Dynamic Toolchains centers *workflow exploration*, demonstrating how automatic data interchange can enable rapid feedback across diverse configurations of data, machines, and materials, such as in computer-controlled watercolor painting.
- The first iteration of KnitScape centers *material exploration*, demonstrating how immediate feedback provided by a yarn simulation can support development of intuition about emergent color patterning effects.
- The second iteration of KnitScape centers *design exploration*, demonstrating how composable design elements enable translation of insights about material

behavior (such as the tendency of knit fabric to curl) into more complex designs (such as self-pleating patterns).

In this chapter, I use insights from my systems work to discuss how tools can support digital/physical creativity by enabling exploration of emergent behavior at the digital/physical divide. I frame this discussion through the practice of *swatching*: a way of reasoning about creative processes by making small artifacts, or *swatches*, which concretely document relationships between designs, techniques, and materials.

Swatching has long been central to traditional craft practices, and it is a powerful method for exploring emergent behavior. I argue that tools for digital/physical creativity should embrace *swatchiness*. To this end, I propose four principles of swatchiness: *rapid insight*, *tinkerability*, *multi-modal exploration*, and *focused support*. These principles provide a framework for developing *swatchy tools*: tools which foreground digital/physical relationships and enable deep engagement with the interplay between design and process.

Swatchiness in Context

Chapter 2 explores the workflows of an online creative community of artists on #PlotterTwitter, contrasting their diverse approaches to digital/physical creativity with the historical evolution of digital fabrication infrastructure. The practices shared on #PlotterTwitter often challenge the dominant model of digital/physical workflows (termed the "canonical workflow"), which imposes a rigid separation between digital designs and physical processes.

A significant part of the creativity we saw on #PlotterTwitter stems from artists' ability to negotiate the physical constraints of machine precision and materials alongside their digital representations. For example, watercolor paint can layer and pool unpredictably when applied by a brush, creating emergent effects that might initially seem surprising or even undesirable. However, by working closely with their materials, artists develop an understanding of these behaviors and incorporate them into their finished artworks. This iterative and deeply contextual style of working is infeasible when digital tools force design and material processes to be considered separately.

The isolation of design and process in the canonical workflow reflects a broader cultural bias in computational and engineering domains, which have traditionally valued abstraction and formalism. As outlined by Turkle and Papert [241] in their essay on *epistemological pluralism*, this can alienate those who prefer concrete and contextual approaches to knowledge-making, such as *bricolage*–a style of working through active negotiation and re-negotiation of well-known materials.

Working closely to their materials enables the bricoleur to respond quickly to the system being constructed, exploring by "playing" directly with the elements of a system and discovering how they interact. Bricolage is an apt description for the majority of workflows shared on #PlotterTwitter, where the "materials" being negotiated include elements on both sides of the digital/physical divide. Sometimes these are literal, physical materials (pens and paper, ink, paints, thread, and sand), but they also could be elements like code, geometric designs, machine settings and motion, and how all of these are configured to construct a workflow.

Bricolage closely parallels the process of swatching, which can be understood as a way of "playing" with relationships within a digital/physical workflow. When swatching, each iteration is captured in a physical artifact—a swatch—that serves as both a record of exploration and a tool for reflection. Susan Guagliumi discusses this iterative, exploratory approach in the introduction to her book *Open Spaces* [77], a compilation of machine knitting techniques communicated through hundreds of swatches. She describes swatching as "*playing the game of what ifs*," an ongoing dialogue between materials and methods:

> "What will happen if I do this six times instead of five? What will happen if I work in two or more colors? Knit an even or odd number of rows between repeats? Use more or fewer stitches per repeat? What if, what if, what if. Those are the words that keep me exploring and experimenting and *excited* about machine knitting."

Guagliumi's excitement to "play the game of what ifs" suggests that she has figured out how to create the conditions for "flow" in her process. As described by Csíkszentmihályi [41], flow is characterized by a deep immersion in an activity, where, among other elements, clear goals, immediate feedback, and a balance between challenge

and skill combine to make the experience intrinsically rewarding. When an activity reaches this state, it becomes "autotelic"—an end in itself.

Swatching aligns naturally with these conditions for flow. Swatches are small, and (importantly) likely faster to produce than final products. This speeds up opportunities to make discoveries and get feedback on decisions, which in turn enables someone to sustain a sense of momentum. Swatching scaffolds creative activity by providing opportunities to define incremental, manageable goals, such as testing how different fabric textures might feel when draped on the skin. These small steps anchor exploration without overwhelming it.

Importantly, swatching is not about ignoring the broader context for creative activity. Rather, it enables someone to iteratively resolve questions about what they *can* make, what they *want* to make, and *how* to make it. This positions swatching as a bottom-up form of exploration; a way of building a "process construction kit" which is composed not of physical parts, but of relationships. Instead of assembling swatches like building blocks, the relationships they represent are used to inform and refine a broader workflow.

This was the style of making which we observed on #PlotterTwitter. While artists spent a significant amount of time exploring emergent material behavior through swatchy, low-level manipulation of digital/physical relationships, they were *also* able to harness insights gained from that process towards the creation of a final artwork. Sometimes, an explicit goal clearly motivated low-level exploration, such as achieving greater dynamic range by layering pens and pencils (documented in Figure 2.8). Other times, the opposite was true: a goal followed exploration. An artist notices the difference in line width when they set their pen to different heights, and wonders whether they can create images by modulating the pen's height while it is moving.

This bottom-up approach to making things also characterizes tinkering, which, like swatching, is not random or aimless. As Resnick and Rosenbaum [199] observe, "true tinkerers know how to turn their initial explorations (*bottom*) into a focused activity (*up*). It is the combination of *bottom* and *up* that makes tinkering a valuable process." Similarly, Victor [251] emphasizes how moving between "levels" of abstraction—not remaining fixed at a single level—is a powerful way to gain insight into a system.

When considering how software tools might support swatchy workflows, it is im-

portant to recognize how tools play significant role in shaping the way we work. The primitives, abstractions, and "black boxes" chosen by the developers of creative tools define which ideas can be explored with those tools [200]. In this way, creative tools establish a "normative ground," shaping how their practitioners think, act, and express themselves [138]. Formalism emphasizes some possibilities, and obscures others.

The CAD/CAM/CNC model exemplifies this: it formalizes workflows to separate the design of an artifact from the physical processes which make it. This encourages a style of making which shines when a well-defined goal lies solidly in the design "space" of a CAD tool and can be easily reached using the language of the tool. Standard interchange formats also aid collaboration and replication across time and space, meaning that the same file can be executed on different machines to get the same thing: a replacement knob for a washing machine can be designed once and printed around the world.

Academic research on digital fabrication places significant value on building tools which conform to this top-down style of organizing digital/physical processes. Often it is stated directly in this work that the motivation behind building new high-level abstractions is to enable users to design without concerning themselves with low-level implementation details (e.g., [169, 152, 112]). However, while digital/physical creativity can and does occur *within* these formalisms, it also occurs *beyond* them. While high-level abstractions are helpful for exploring design spaces which are well established and understood, we also need tools that *help* us be concerned with low-level details: we need swatchy tools.

"Concern" with low-level implementation details is precisely what enables many new discoveries, and this is demonstrated across practice and academia. For example, 3D printed structures such as bridges, blobs, and textiles cannot be designed in traditional CAD tools, but they can if someone is able to manipulate G-code commands directly [227, 188, 60]. This kind of creative exploration is only possible when working directly with low-level behavior.

The dominant value placed on top-down approaches to digital/physical creativity risks sidelining other valid styles of creative work. Swatchy tools complement abstraction-centric approaches by helping people engage with low-level details of a digital/physical process. This is what makes swatchy tools powerful: by helping us ex-

plore and understand emergent behavior, they effectively help us design *beyond* the limitations of existing abstractions and start to articulate new ones.

Practical Swatching

The reasons why someone might create a swatch vary widely. However, it is clear that the fundamental activity of swatching is important across diverse creative workflows and contexts, some examples of which are described in this section.

To experiment

Swatching is often a method of experimentation: when we lack a robust understanding of a process, we can create swatches to answer questions about emergent behavior. For example, the performance of weave structures integrated with elastic threads is difficult to predict, and creating test swatches is a way to explore this interaction [48].

We often see academic researchers create numerous small samples to evaluate or demonstrate the design space of a novel process or material, for example, by varying printer parameters and infill density when electrospinning [203], demonstrating the effects of material composition of 3d-printed play-dough [29], or defining the bounds of a recommended parameter range when using under-extrusion to create 3D-printable textiles [60]. Each of these are examples of design spaces that can't be explored with traditional tools and methods, but can be explored when someone is able to manipulate behavior at a low level.

To inform next steps

Swatching is a way to collect specific information which will inform design and implementation: information about material performance can impact design decisions, and a designer may not be able to confidently proceed without it. For example, the dimensions of knitted fabric vary dramatically depending on yarn type, stitch pattern, tension, stitch size, and treatment (i.e., washing and steaming). When designing something to particular dimensions (such as most clothing), it is necessary to first know the fabric gauge (stitches and rows per inch) in order to specify how wide a piece should be in stitches. Knitting, washing, and measuring a *gauge swatch* is one established

method for gathering this information. In most cases, designing and knitting a whole garment just to take measurements would be impractical and likely wasteful, especially for small-scale and one-off production. Here, swatching is a way of isolating the variables of interest and extracting relevant information to inform next steps.

To extrapolate

Humans are capable of extrapolating from small swatches. For example, we might hold up paint swatches to the bedroom wall when picking a new paint color. Someone might apply swatches of eyeshadow on their forearm in order to survey a new palette. Swatches embed a process within a physical context, providing a tangible reference point to the physical world which is far easier to extrapolate from than a purely digital representation.

To document

Swatches are typically not intended to be disposable, and the information they hold can be useful beyond the original process which led to their production. A swatch originally made for one purpose, such as determining the stitch and row gauge of a yarn for a new sweater, can be reused to make a hat with the same yarn. It is common across makerspaces to see material benchmarks for different machines, for example, an overhang or heat tower sitting next to a 3D printer, or a kerf test next to a laser cutter. These kinds of swatches demonstrate exploration of behavior that is inherently contextual and cannot be abstracted.

While 3D printers and laser cutters and their associated materials often come with recommended settings for their operation, actual results can vary widely. Producing dedicated swatches to document these effects is important because “finished” pieces (a sweater, a game piece, a laser cut box) will likely leave the place of their production. A swatch instead provides a persistent process and material reference which might lose its meaning if removed from its production context.

To learn

Swatching is an excellent way to learn new techniques and develop an associated skillset. The smaller scope of a swatch enables someone to focus on an isolated technique without worrying about “messing up” a finished project or needlessly wasting

materials. Through swatching, someone can practice and build confidence before moving on to a larger project, lowering the stakes of taking the time to work out unknowns.

To communicate

Swatches can provide a common reference when discussing materials and processes. Swatches have been proposed as an academic contribution, where they are described as the "partial solutions which make our projects possible" [187]. Because swatches are smaller in scope, it is also likely that they more easily transported than their finished counterparts, and compiled into a collection. Swatches foreground relationships within a process which may be less obvious if presented in a final product, enabling communication to center on process over product.

To enjoy ourselves

Creativity is often an autotelic activity, meaning we do it because it is inherently enjoyable [38]. Swatching can also be a way of making something when you just want to make something; it does not need to be driven by the goal of producing a complete, functional artifact. It can be performed on its own to "play around" and tinker with different techniques and materials, explore new ideas, or practice and refine skills.

Principles of Swatchiness

What makes a creative tool a *swatchy* tool? Swatchy tools help people compose designs and artifacts, but they also help people reason about broader creative processes by emphasizing the *relationships* which lead to emergent behavior at the digital/physical divide.

This section outlines four "Principles of Swatchiness" (summarized in Table 6.1) which I have developed through insights gained from my time building the systems described in prior chapters. I hope these principles will help other developers think critically about how their tools support process-level reasoning and exploration of relationships within digital/physical workflows.

Principle	Description
Rapid Insight	Swatchy tools should aim to help someone quickly develop intuition about digital/physical relationships by minimizing the time it takes to move between idea, outcome, and intervention.
Tinkerability	Swatchy tools should encourage playful, undirected engagement by making designs easy to modify and rewarding curiosity with immediate visual or real-world feedback.
Multi-modal Exploration	Swatchy tools should support exploration of multimodal relationships by providing side-by-side access to diverse parameters and feedback.
Focused support	Swatchy tools should focus on supporting a narrow range of creative tasks over access to a broad design space.

Table 6.1 *Four Principles of Swatchiness*

Rapid Insight

A guiding philosophy of the pioneering mathematician and computer scientist Richard Hamming was "The purpose of computing is insight, not numbers" [80]. Hamming elaborates that this doesn't simply mean that the numbers we get out of the computer are read and used—it also means that the choice of algorithms and techniques influences not only how we get results, but how we interpret them. He writes, "computing is, or at least should be, intimately bound up with both the source of the problem and the use that is going to be made of the answers—it is not a step to be taken in isolation from reality."

Similarly, tools for digital/physical creativity deeply influence how we approach the process of making something, which in turn influence how we interpret the process we are engaging with. Someone who has only used tools which cast 3D printing as a way to print a solid object out of planar layers may take a very long time to realize that their 3D printer can make a much broader space of interesting structures (e.g., [227, 188, 60]). This tradeoff between ease of use and expressivity is inherent to abstraction, and what motivates prior recommendations to be *very careful* when choosing black boxes in creative tools and toolkits [200, 75]. While tool developers should indeed be careful, this may also frame unsupported explorations as ones which we haven't found the right black boxes for yet.

When building tools for digital/physical creativity, we shouldn't just be trying to choose the "best" black boxes—the ones that balance expressivity and ease of composition with a large design space. We should also recognize that digital/physical creativity takes place across many different "levels" of abstraction. In the absence of higher-level abstractions which allow us to directly specify the things we want to make, *we can still explore possibilities* by developing intuition about interactions within a system that we can use to guide our explorations [251]. In this way, supporting *rapid insight* is how swatchy tools complement, rather than supplant, an abstraction-centric approach to tool development.

The first iteration of KnitScape takes a swatchy approach to supporting the design of slip and tuck colorwork: rather than trying to abstract this colorwork process so that patterns can be specified directly, KnitScape enables someone to explore how color patterns emerge from interactions between stitches and yarn sequence. An abstraction-centric approach to supporting this same design problem might have involved choosing a "higher level" design representation, perhaps vector paths and curves which someone could use to directly draw their desired output pattern, and figuring out how to transform that representation into a stitch pattern and yarn sequence which would faithfully produce the desired color pattern when knit.

When sharing KnitScape, how to support this "inverse" problem is very often one of the first questions I am asked by people who come from engineering and/or academia, which calls back to the value placed on abstraction and formalism in traditional engineering domains [241]. I do not think it is wrong to want to build higher-level abstractions, and I think a tool which enables this kind of direct specification would be valuable. But, crucially, higher-level abstractions should not be the *only* point of access to digital/physical creativity. We need both: abstractions for navigating defined design spaces, and swatchy tools which help us develop insight into processes by thinking explicitly about the underlying relationships ans interactions.

Insight isn't only helpful when we're working outside of defined behavior—it is also important for understanding abstractions and how they are embedded in the broader context of a digital/physical workflow. For example, people experienced with 3D printing understand how orientation, overhangs, infill, and support material influence the quality of a resulting print, and are able to incorporate insights about those relation-

ships into their designs. However, import/export bottlenecks slow down iteration cycles and obscure these digital/physical relationships, which can make them frustrating for newcomers to digital fabrication [90, 175].

Swatchy tools should try to help someone develop intuition about relationships as quickly as possible, by minimizing the time between idea, intervention, and refinement. For digital/physical creativity, this also means finding ways to make machine execution as immediate as possible so that someone can maintain a correspondence between design and outcome. If direct control of a machine is not realistic (which certainly might be the case), the tool could try to export files which need no intermediate processing (e.g., by a tool such as a slicer).

An important part of designing with this principle is realizing that if given the chance, human intuition can make up for gaps in our technology. Creative exploration and discovery spans "levels" of abstraction, and we can and should take advantage of human insight when building tools for digital/physical creativity. Swatchy tools should initially focus on complementing, rather than displacing, natural human abilities. Things which can be reasonably intuited by humans (such as material behavior via accurate physics simulations) should be lower priority for developers than things which can *only* be improved by software (such as file format wrangling or immediate machine execution).

Tinkerability

Emergent behavior is not specified directly; rather, it *emerges* from interactions within a system. Exploring emergent behavior is therefore a process of exploring cause and effect. It needs to be easy to "mess around" with the elements of a system by making small changes and observing the impacts of those changes, a style of engaging with a problem that characterizes tinkering [199]. Tinkering has long been a way people explore emergent behavior in craft practices, and digital/physical creativity is no different. Swatchy tools should try to be highly tinkerable: they should encourage playful, low-stakes experimentation at different points in a digital/physical workflow, and reward curiosity with quick visual or real-world feedback.

Dynamic Toolchains uses automatic data interchange between modules to enable

immediate feedback across the digital/physical divide. Toolchains can be configured in which changes made to a digital design (such as a modification to a brushstroke path) can produce immediate effects in the physical world (the plotter moving the brush). The short time between intervention and outcome helps someone develop insight into the interaction between on-screen path and paintbrush motion, but it also invites hands-on, playful engagement with different aspects of the digital/physical process. This enables someone to "play the game of what ifs", as described by Susan Guagliumi.

A dynamic toolchain for watercolor painting encourages engagement at different points in the digital/physical process. On the digital side, someone can tweak or draw a brushstroke path, but they can also easily change the path repetition or move the brush around the canvas. On the physical side, they can change the position of the brush in the tool holder or the amount of paint on the brush. Even when changes do not produce immediate effects on the physical side, they are immediately previewed on the toolchain canvas. The painting toolchain is tinkerable because it invites someone to "mess around" with the elements of a particular digital/physical workflow without thinking about the logistics of how different pieces are connected.

Watercolor brush strokes are very fast to execute on a plotter, making it an ideal context for exploring immediate feedback across the digital/physical divide. However, many other fabrication processes require significant execution time, or significant time setting up the machine or materials, making it logistically challenging to implement immediate feedback in the same way as the painting toolchain.

Machine knitting is one example of such a process. KnitScape also promotes a tinkerable style of engagement with a digital/physical workflow, but instead relies on visualization to provide immediate feedback on design decisions (rather than machine execution). Rather than specifying a desired colorwork pattern directly, small changes to the stitch pattern or yarn sequence immediately regenerate the yarn topology and visualization, showing how they will interact to produce the colorwork pattern. This means that someone does not need to have a clear idea of what color pattern they want to design when using KnitScape. However, with practice, someone can also learn to use KnitScape to design pre-determined patterns like the "Numbers" swatch shown in Figure 4.13.

KnitScape adds another layer of play to this bottom-up engagement with knit colorwork via the physics simulation, which provides further payoff by animating how the fabric will deform under the fabric topology someone has designed. The simulation, which is satisfying and often surprising, is—by far—the favorite part of the tool of everyone I have shown it to, regardless of age or background. "Do it again" is the common phrase uttered by my five-year-old cousins, researchers visiting my demo booth at an academic conference, and engineers at a hacker convention. Beyond helping us evaluate our progress, observing cause and effect also compels us to ask "why" something happens, encouraging us to investigate phenomena we do not understand.

People should be able to use swatchy tools to tinker with digital/physical relationships and "play the game of what ifs", making small, incremental adjustments that lead to serendipitous discoveries. By prioritizing low-stakes exploration, these tools promote bottom-up understanding of complex tasks through hands-on engagement. Tinkerable tools should try to enable small changes to be made (and the results of those changes observed) without a concrete plan, goal, or understanding of the broader system.

Multi-modal exploration

Exploration is an essential part of creative processes. Accordingly, supporting exploration is seen as perhaps the most foundational way to support creativity in software tools [59, 220, 200]. Prior recommendations emphasize supporting exploration via ease and speed of composition: making designs easy to compose and modify, straightforward to backtrack and compare alternatives, and fast to "sketch" lower-fidelity results.

However, digital/physical creativity crosses the digital/physical divide, which means that by definition, the "space" of exploration is never confined to an individual digital tool. Any exploration which occurs in a digital tool *will have a downstream effect* - whether it be to other digital tools, or with physical processes. This can be challenging to people who do not fully understand the relationship between design and process [90], but exploring emergent behavior at the digital/physical divide is also important to practitioners developing novel creative workflows and tools [244, 35, 85].

To prior recommendations to support exploration, I add an explicit emphasis on *multi-modality* for tools for digital/physical creativity. The principle of multi-modal exploration emphasizes that it is important to be able to intervene and evaluate the results of exploration across diverse contexts. Tools for digital/physical creativity need to not only help people explore *within* their design space, but *beyond* it.

Enabling multi-modal exploration was a core motivation behind the development of Dynamic Toolchains, which supports exploration of artifacts in the conventional sense by enabling customization of individual modules and composition of toolchains from modules. However, a key idea explored in Dynamic Toolchains was that immediate feedback *between tools* and *across the digital/physical divide* enables a user to explicitly focus on emergence by being able to intervene and observe results across tools and contexts. By speeding up iteration loops by eliminating import/export bottlenecks, exploration in Dynamic Toolchains became multi-modal.

This means that the immediacy offered by Dynamic Toolchains foregrounds exploration of *relationships* across the digital/physical divide. For example, the watercolor toolchains couples a diverse parameter set (including a drawing pad and SVG patterning and translation tools) with multiple forms of visualization, immediate machine execution, and a human-in-the-loop. This connection made it possible to explore and understand the relationship between an input brushstroke path and the effect of dripping watercolor paint on the brush.

Multi-modal exploration is powerful because it enables us to probe relationships which cannot be represented directly in existing abstractions. As another example, even though the KnitScape instruction chart does not visually resembles slip and tuck color patterning, KnitScape still enables exploration of slip and tuck colorwork via multi-modality. The immediacy of the yarn visualization reveals the relationship between yarn sequence and stitch pattern, making it possible to design patterns like the "numbers" swatch in Chapter 4.

In order to enable exploration of complex digital/physical relationships, swatchy tools should integrate multi-modal inputs (e.g., design geometry, machine execution parameters, material properties) and outputs (e.g., visualization, simulation, and real-world execution). Side-by-side control of diverse input parameters, coupled with visualization and/or immediate execution, should facilitate a holistic understanding of how

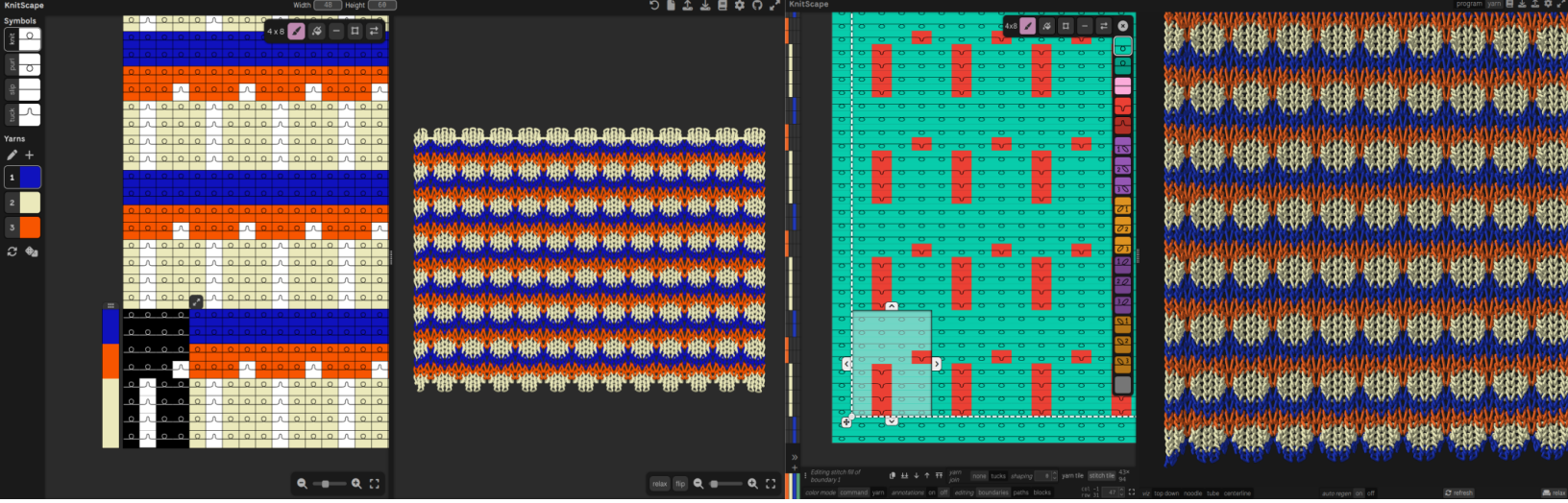


Figure 6.1 *Everything which can be designed in the original version of KnitScape (left) can be designed in the second version (right). However, the original KnitScape foregrounds the relationship between stitch pattern and yarn color sequence, making it easy to explore the resulting emergent behavior. Features added to the second version which make it easier to compose more complex designs ended up obscuring this relationship.*

diverse elements interact to produce emergent effects.

Focused Support

The final principle of *focused support* is meant to prompt reflection from the tool developer and contribute some nuance to prior literature on design principles for creativity support. Specifically, the principle of "low threshold, high ceilings, and wide walls" [200], which recommends tools try to make it easy to get started while also providing many ways of exploring a broad, diverse design space. To this I will add that swatchy tools should help the user *focus* on relationships of interest.

The second iteration of KnitScape can do everything the original version can and more. However, even though KnitScape II could be used to design the same patterns as KnitScape I, I found myself opting for the original version when introducing people to my research. I originally thought that this meant I had significantly raised the "threshold" of KnitScape II–and that I should focus on trying to lower it. However, on reflection I realized that the underlying problem was that some of my interface changes had ended up obscuring the *relationship* which prompted the development of the original KnitScape: the interaction between a yarn color sequence and a stitch pattern (Figure 6.1).

Swatchy tools should not only help someone explore design spaces, but also fo-

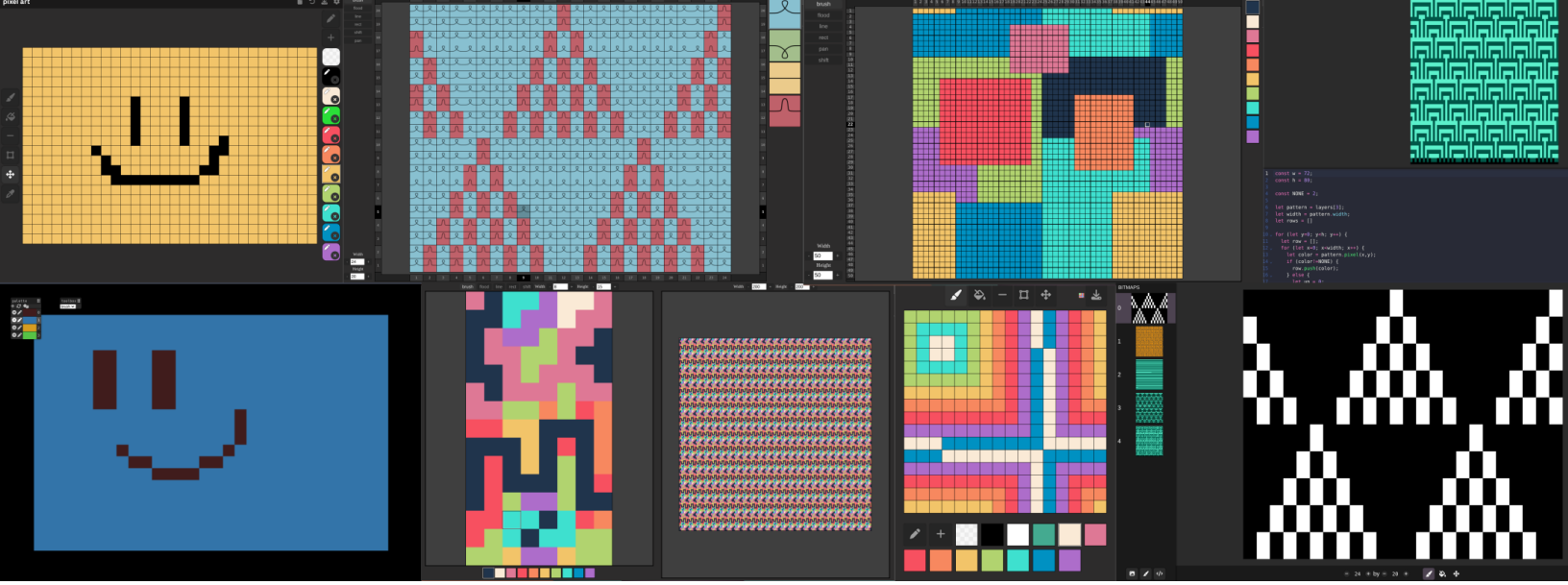


Figure 6.2 *Swatching can take many forms. I explored different aspects of KnitScape's implementation by creating many different small bitmap editing tools. I learned something from each of these, and still revisit them to copy and paste bits of code into other projects.*

cus in on and explore relationships within that design space. To accomplish this, the tool should expose (or enable someone to easily isolate) a particular relationship or parameter of interest, ideally without adding too much interface complexity such as complex selection states. One way to address this might be to add additional editing contexts which surface curated parameter spaces, similar to the multilayer interfaces suggested by Shneiderman [220]. For example, the second iteration of KnitScape could offer a fabric structure design context where everything to do with the shaping of panels is hidden, and only editing tools for designing fabric structure are visible. Such an interface would enable someone to focus on iterating only on fabric structure without being distracted (or overwhelmed) by unnecessary tools. This is one example I can imagine for focused support which could extend to other domains as a fruitful area of future research.

Swatchy Mindsets

When I stepped onto the Olin campus, I was suddenly surrounded by tools, but I was not suddenly empowered to make almost anything. It took practice, and through practice I learned to work with my new tools and use them to shape the world. The project-based curriculum at Olin was devoted to helping us learn to practice independently by giving us structured time, space, and resources to first practice by exploring our own

interests.

My time at Olin helped me develop what I call a "swatchy mindset:" an inclination to approach creative work through focused, iterative exploration via small artifacts. To me this feels like the most natural way to engage with new ideas, and I often do so unconsciously. For example, when figuring out implementation details of KnitScape I built an array of bitmap editors (Figure 6.2). This gave me an accessible survey of techniques to draw from when building more complex systems. I did not plan this approach ahead of time, rather, each small project informed the next, shaping my understanding of what I wanted to build and how I was going to build it.

Of course, iterative exploration via testing, prototyping, sampling, or swatching is widely seen as a critical creative strategy, and it is strongly emphasized in engineering and design programs around the world. My point is that swatchy mindsets are developed through practice. We like to idealize our processes, especially when recounting our work to others. It is tempting to downplay the iterative and exploratory aspects of creativity in favor of coherent narratives which trace ideas to outcomes [240], when in reality processes have many "loose ends" [73].

I have worked in undergraduate engineering classrooms for eight years, first as a teaching assistant and later as instructor of record. Through this I have spent hundreds of hours working directly with students, and I have observed that many students do not naturally decompose large problems into small pieces. For example, many students try to build a complete circuit before testing it, or write multiple functions without running their code. Some students consider "starting simple" to be an inefficient way of building things. Educational programs try to help students develop swatchy mindsets by providing a structured environment in which they can see the value of iteration through practice.

However, I think we can also encourage swatchy mindsets by building swatchy tools. Swatchy tools can make the process of iterative exploration engaging, rewarding, and addicting by tightening the loop between cause and effect. By encouraging us to maintain collections of past artifacts, swatchy tools can help us find the motivation to keep going by making it easy to survey our progress. By making our processes more visible to us, swatchy tools can help us see value in the parts of creativity which might be otherwise hidden behind a desire to reach an end product. A swatchy mind-

set represents a general approach to creative work, and its value therefore transcends any specific domain. This is important for digital/physical creativity because it is inherently deeply interdisciplinary. Creative success therefore hinges on the capacity to meaningfully connect ideas across digital/physical domains.

In *The Art of Doing Science and Engineering* [79], Richard Hamming writes, "Once you get your courage up and believe that you can solve important problems, then you can. If you think you can't, almost surely you are not going to." I agree with Hamming. To have creative agency you need more than access to powerful tools. You also need to believe in yourself. You need to have confidence in your ability to focus and direct your powerful tools, and with them reach a satisfying outcome. I think that swatching is one of the best ways to develop this confidence through practice, and I think swatchy tools for digital/physical creativity can encourage a commitment and dedication to practice. Ultimately, I think swatchy tools will not only help us make things today, but also help us imagine what we *could* make tomorrow.

Chapter 7

Conclusion

This dissertation documents four projects through which I explore how software can support creativity across the digital/physical divide. From studying #PlotterTwitter to developing Dynamic Toolchains and two iterations of KnitScape, I have examined how software can help people engage with emergence as a way of building intuition about digital/physical relationships. By focusing on swatching as a practice for reasoning about a broader creative process, these systems demonstrate how playful, bottom-up exploration can complement top-down abstraction. In this conclusion, I outline future directions for this research and close with a personal reflection on how my work, particularly KnitScape, has shaped my perspective and practice.

Future work

I see three main avenues where future work might expand on the ideas and systems presented in my dissertation: exploring how bottom-up exploration supported by swatchy tools might integrate with top-down approaches common in today's tool ecosystems; bringing swatchy tools to new domains beyond the ones I have explored here, particularly textile crafts; and working to bring KnitScape to a broader audience.

Bottom-up meets top-down

In the previous chapter, I explored how swatchy tools enable bottom-up exploration of digital/physical processes, offering an approach that complements—rather than

replaces—top-down abstraction. Central to this discussion is the recognition that both approaches represent valid and valuable styles of making, each with its strengths depending on the context. This suggests a key question which may guide the design of future tools: how might we better support *transitions* between low-level, iterative exploration and higher-level, goal-directed activities?

The second iteration of KnitScape demonstrates one possible direction via *composability*. Higher-level abstractions (e.g. stitch paths and fills) are built from lower-level primitives (stitch patterns and vector paths). This in turn enables vertical movement between exploration of low-level behavior and higher-level designs. Composability in KnitScape is aided by the compatibility between its visual representations: vector paths and raster stitch patterns are both easily expressed in a shared visual environment. There are other processes where this translation will be more complex. For example, if someone is able to prototype a way of building forms through stacked blobs using a tool like p5.fab [227], how might they apply that technique to a 3D model created in a CAD environment?

Bridging the gap between emergence and abstraction is not just about transferring data between domains, but about helping someone maintain momentum when moving between different levels of representation. I think that tools which facilitate these transitions have the potential to be among the most powerful for digital/physical creativity.

Swatchy tools for other domains of textile craft

Swatchy tools could be built to support many domains of digital/physical creativity, with machine knitting serving as just one example of a process where emergent behavior arises from low-level interactions. My personal interests lie in continuing to build swatchy tools for other textile crafts, such as weaving, crochet, and braiding. As an avid crocheter, I am particularly interested in building an editing environment for crochet patterns and tackling the associated challenges, such as generating topology for a yarn visualization from crochet pattern notation, which might come with some unique challenges due to the freeform nature of crochet.

My work simulating and rendering yarn behaviors offers a foundation for this

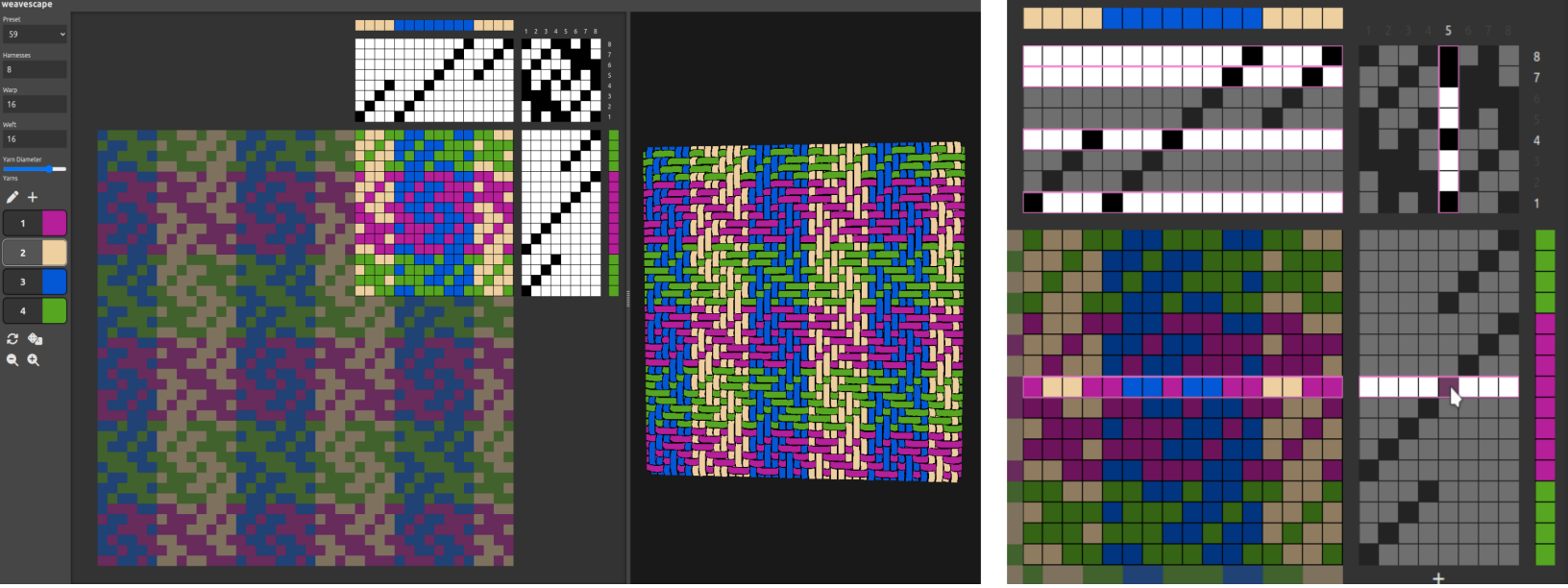


Figure 7.1 *WeaveScape (left) is a swatchy tool for designing weaving drafts. The WeaveScape editor also visualizes the relationship between treadling, tie-up, and threading (right).*

future exploration, as demonstrated by development of "WeaveScape," a swatchy tool for visualizing weaving drafts (Figure 7.1) built in collaboration with Shelby Wilson [1]. We were able to use a number of elements from KnitScape (such as the yarn renderer and sidebar) with little modification when building WeaveScape. The hosted site is available at `https://weavescape.net/` and the source code is located at `https://github.com/branchwelder/weavescape`.

KnitScape (Out There)

While KnitScape and other tools I've developed are open source, they make many assumptions about the contexts in which they will be used—assumptions shaped by my own development environment and preferred style of working with a manual knitting machine. I would like to focus my future work on KnitScape on accommodating broader audiences by adapting it for hand knitting. Aspirationally, I also see continued development of KnitScape (and related tools) as a path toward revival of domestic machine knitting.

HandKnitScape

Hand knitting presents unique challenges and opportunities distinct from machine knitting. A hand-knitting version of KnitScape would need to incorporate charting no-

[1] `https://shelby.cool/`

tation similar to traditional hand-knit charts. KnitScape already supports a significant portion of hand knitting patterns, but extending this to a general charting and simulation environment for hand knitting will be significant. Hand knitting allows for far more variety in fabric topology than machine knitting, and incorporating support for these techniques into the yarn visualization will need to happen case-by-case. Additionally, knitting patterns are typically not exclusively communicated via charts, and patterns which include charts often also rely contextual information and accompanying explanations in text.

Where does a tool like KnitScape fit into hand-knitting workflows, and how can it provide meaningful support for such open-ended practice? A pragmatic starting point might be not a generic design interface, but a fabric structure editor for designing pattern repeats. Such a tool could explore how users could define custom operations which are linked to simulation parameters—for instance, creating a symbol that represents a loose-tension stitch. Even small steps in this direction could fill a significant gap, given the limited number of tools currently available for design of hand knitting patterns which include any form of visualization or simulation.

Reviving domestic knitting machines

Domestic knitting machines were once widely accessible but struggled to transition to the digital era. KnitScape could help reintroduce these machines to modern audiences, making machine knitting more approachable and less intimidating. A possible starting point might be to build a tool which matches the capabilities of circular sock machines or plastic machines such as the Sentro or Addi. While these machines are limited to producing striped tubes and shaped sheets, their design space is rich enough to have fostered active online communities dedicated to exploring every possibility, and recent efforts to automate color patterning on these machines suggest that the community is committed to exploring their potential.

Adapting KnitScape's visualization to show the step-by-step process of knitting a pattern on a manual machine could further lower the barrier to entry. From my experience learning to use a knitting machine, one of the biggest hurdles was understanding how fabric topologies were formed by the machine, such how action of latch needles or how carriage settings changed the needle actuation. Future iterations of KnitScape could include animated simulations that visually demonstrate these processes. I have

already begun work in this direction by building a custom control board and interface for the Silver Reed SK840 which could enable direct control from KnitScape. I also hope to explore how this work might integrate with the existing AYAB project for controlling Brother machines [13].

Reflection

I have always found creativity to be a deeply rewarding process, driven by the simple pleasure of making something new. Being able to build tools which support my own creativity has only enhanced that feeling. While I am proud of my other projects, KnitScape, in particular, was a turning point for me. My growing fluency with browser technologies helped me see progress quickly and sustain a sense of momentum during development. It was when working on KnitScape that I began to formalize the ideas behind swatchiness, making it not just a creative tool but a foundation for the broader themes of my dissertation.

I began working on KnitScape with the simple desire to design and knit my own things. Although I had access to automatic machines (and have since learned how to use them), they were overwhelming when I was first getting started. The manual Silver Reed knitting machine felt accessible and tangible, and I was amused by its archaic interface and the way it folded up into its own carrying case. But as I attempted larger projects, I quickly found myself frustrated with how there were no tools available that offered meaningful visualizations to help me understand the relationship between patterns and outcomes. Without this, I was worried that I would waste my time and resources or design things I didn't like. At first, this hesitation made it difficult for me to get started, but swatching gradually became my way of managing these unknowns and navigate the trepidation I was experiencing.

Building KnitScape has enabled me to engage every part of my brain. Using the manual knitting machine is satisfying and meditative. Designing and building interfaces, writing code, and watching myself improve—all while using these tools to create—has been deeply rewarding. I have found that domains which didn't initially capture my interest, such as computer graphics and simulation, have become fascinat-

ing when paired with this problem space. Through the process of building KnitScape, I discovered not only the joy of engaging deeply with technical challenges but also a connection to the rich, and sometimes elusive, history of machine knitting.

When I was learning to use the Silver Reed I often felt like an internet archaeologist. I unearthed blogs and email chains where the last posts were made in 1990s. I bookmarked links to patterns which stopped working. I read about fancy attachments for knitting machines, only to find they stopped being manufactured decades ago, and I found that even for my machine, replacement parts were almost impossible to source. A few months into my time working on KnitScape, Susan Guagliumi, the respected author of multiple books on machine knitting who described swatching as "playing the game of what ifs," retired and closed her website.

At times it felt like I had stumbled onto a craft which had survived (and thrived) for nearly 200 years before I was born, just in time to watch it fade away. But this isn't the whole story.

While some resources have faded, others have simply evolved. Communication has shifted away from personal blogs and email chains to 25,000-person-strong groups of passionate manual machine knitters on Facebook. Through these groups, I learned of projects building extensive archives of machine knitting fashion magazines, manuals, and pattern books [2, 1], and spent late nights giggling at wacky designs from the 70s. I heard excitement about new books being published on manual machine knitting techniques, and I bought them all. I discovered classes through The Knitting School Online (TKSO) and immediately signed up. I've built a long list of active YouTube channels producing high-quality tutorials. Videos of people creating on inexpensive circular machines regularly receive hundreds of thousands of views on Tik Tok. The AYAB project, an open-source controller for Brother machines, has garnered renewed interest, and I've seen numerous recent projects from makers designing 3D-printed circular machines. This community is finding new ways to grow, and I've enjoyed watching it take on new life.

KnitScape has also enabled me to engage with people in new and unexpected ways. Early on I decided to keep a small bag of swatches in my backpack, and I have noticed that they invite interaction: people instinctively reach out to rub, stretch, and feel their textures. When I took the Silver Reed to a hardware hacking conference and set it up

next to a wire bonding machine and scanning electron microscope, I fielded nonstop questions about how it worked and where to get one. I realized that while most people don't know the details of knitting (and call the knitting machine a "loom"), almost everyone knows someone who knits. Often, this person is someone dear to them: a close family member, friend, or partner. Because of this, a polite question about what I do in graduate school frequently turns into someone telling me a story about a person they care about once they learn what I'm working on. Sometimes, I'll run into an avid knitter eager to share photos of their latest projects.

KnitScape remains a deeply personal project—one I would continue working on even if I were its only user. But experiencing the enthusiasm of others and discovering the deep personal connections people have to this domain has been a source of immense motivation for me. It has transformed KnitScape from a solitary pursuit into something that connects me to a vibrant community. I am excited to continue this journey and to see where it leads, not just for myself but for all those who find joy in the process of making.

Bibliography

[1] Machine Knitter's Treasure Chest. https://needlesofsteel.org.uk/index.html, .

[2] Machine Knitting etc. https://mkmanuals.com/, .

[3] Roland Aigner, Andreas Pointner, Thomas Preindl, Patrick Parzer, and Michael Haller. Embroidered Resistive Pressure Sensors: A Novel Approach for Textile Interfaces. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, pages 1–13, New York, NY, USA, April 2020. Association for Computing Machinery. ISBN 978-1-4503-6708-0. doi: 10.1145/3313831.3376305.

[4] Roland Aigner, Mira Alida Haberfellner, and Michael Haller. Loopsense: Low-scale, unobtrusive, and minimally invasive knitted force sensors for multi-modal input, enabled by selective loop-meshing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*, CHI '24, pages 1–17, New York, NY, USA, May 2024. Association for Computing Machinery. ISBN 9798400703300. doi: 10.1145/3613904.3642528.

[5] Lea Albaugh, Scott Hudson, and Lining Yao. Digital Fabrication of Soft Actuated Objects by Machine Knitting. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–13, New York, NY, USA, May 2019. Association for Computing Machinery. ISBN 978-1-4503-5970-2.

[6] Lea Albaugh, Scott E. Hudson, Lining Yao, and Laura Devendorf. Investigating Underdetermination Through Interactive Computational Handweaving. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, pages 1033–1046, New York, NY, USA, July 2020. Association for Computing Machinery. ISBN 978-1-4503-6974-9. doi: 10.1145/3357236.3395538. URL http://doi.org/10.1145/3357236.3395538.

[7] Lea Albaugh, James McCann, Scott E. Hudson, and Lining Yao. Engineering

Multifunctional Spacer Fabrics Through Machine Knitting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, pages 1–12, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8096-6.

[8] Lea Albaugh, Scott E Hudson, and Lining Yao. An Augmented Knitting Machine for Operational Assistance and Guided Improvisation. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–15, Hamburg Germany, April 2023. ACM. ISBN 978-1-4503-9421-5. doi: 10.1145/3544548.3581549.

[9] Lea Albaugh, Scott E Hudson, and Lining Yao. Physically Situated Tools for Exploring a Grain Space in Computational Machine Knitting. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14, Hamburg Germany, April 2023. ACM. ISBN 978-1-4503-9421-5. doi: 10.1145/3544548.3581434.

[10] Celena Alcock, Nathaniel Hudson, and Parmit K. Chilana. Barriers to using, customizing, and printing 3d designs on thingiverse. In *Proceedings of the 19th International Conference on Supporting Group Work*, GROUP '16, page 195–199, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342766. doi: 10.1145/2957276.2957301. URL https://doi.org/10.1145/2957276.2957301.

[11] Kristina Andersen, Ron Wakkary, Laura Devendorf, and Alex McLean. Digital crafts-machine-ship: creative collaborations with machines. *Interactions*, 27(1): 30–35, 2019.

[12] Cecilia R. Aragon, Sarah S. Poon, Andrés Monroy-Hernández, and Diana Aragon. A tale of two online communities: fostering collaboration and creativity in scientists and children. In *Proceedings of the seventh ACM conference on Creativity and cognition*, C&C '09, pages 9–18, New York, NY, USA, October 2009. Association for Computing Machinery. ISBN 978-1-60558-865-0. doi: 10.1145/1640233.1640239. URL http://doi.org/10.1145/1640233.1640239.

[13] AYAB Developers. AYAB - all yarns are beautiful, 2024. https://ayab-knitting.com/, accessed 2024.

[14] Emmanuel Ayodele, Syed Ali Raza Zaidi, Jane Scott, Zhiqiang Zhang, Maryam Hafeez, and Des McLernon. The Effect of Miss and Tuck Stitches on a Weft Knit Strain Sensor. *Sensors*, 21(2):358, January 2021. ISSN 1424-8220. doi: 10.3390/s21020358.

[15] Jeffrey Bardzell, Shaowen Bardzell, and Austin Toombs. "now that's definitely a proper hack": Self-made tools in hackerspaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 473–476, New York, NY, USA, April 2014. Association for Computing Machinery. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557221.

[16] Patrick Baudisch, Arthur Silber, Yannis Kommana, Milan Gruner, Ludwig Wall, Kevin Reuss, Lukas Heilman, Robert Kovacs, Daniel Rechlitz, and Thijs Roumen. Kyub: A 3d editor for modeling sturdy laser-cut objects. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300796. URL https://doi.org/10.1145/3290605.3300796.

[17] Gabrielle Benabdallah, Sam Bourgault, Nadya Peek, and Jennifer Jacobs. Remote Learners, Home Makers: How Digital Fabrication Was Taught Online During a Pandemic. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pages 1–14, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8096-6. doi: 10.1145/3411764.3445450.

[18] G Blauvelt, T Wrensch, and M Eisenberg. Integrating craft materials and computation. *Knowledge-Based Systems*, 13(7):471–478, December 2000. ISSN 0950-7051. doi: 10.1016/S0950-7051(00)00063-0. URL http://www.sciencedirect.com/science/article/pii/S0950705100000630.

[19] Yvan Bonnassieux, Christoph J Brabec, Yong Cao, Tricia Breen Carmichael, Michael L Chabinyc, Kwang-Ting Cheng, Gyoujin Cho, Anjung Chung, Corie L Cobb, Andreas Distler, Hans-Joachim Egelhaaf, Gerd Grau, Xiaojun Guo, Ghazaleh Haghiashtiani, Tsung-Ching Huang, Muhammad M Hussain, Benjamin Iniguez, Taik-Min Lee, Ling Li, Yuguang Ma, Dongge Ma, Michael C McAlpine,

Tse Nga Ng, Ronald Österbacka, Shrayesh N Patel, Junbiao Peng, Huisheng Peng, Jonathan Rivnay, Leilai Shao, Daniel Steingart, Robert A Street, Vivek Subramanian, Luisa Torsi, and Yunyun Wu. The 2021 flexible and printed electronics roadmap. *Flexible and Printed Electronics*, 6(2):023001, June 2021. ISSN 2058-8585. doi: 10.1088/2058-8585/abf986.

[20] Michael Bostock, Vadim Ogievetsky, and Jeffrey Heer. D^3 Data-Driven Documents. *IEEE Transactions on Visualization and Computer Graphics*, 17(12): 2301–2309, December 2011. ISSN 1941-0506. doi: 10.1109/TVCG.2011.185.

[21] Samuelle Bourgault, Pilar Wiley, Avi Farber, and Jennifer Jacobs. Coilcam: Enabling parametric design for clay 3d printing through an action-oriented toolpath programming system. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 978145039421. doi: 10.1145/3544548.3580745. URL https://doi.org/10.1145/3544548.3580745.

[22] David E. Breen, Donald H. House, and Michael J. Wozny. Predicting the drape of woven cloth using interacting particles. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '94, page 365–372, New York, NY, USA, 1994. Association for Computing Machinery. ISBN 0897916670. doi: 10.1145/192161.192259. URL https://doi.org/10.1145/192161.192259.

[23] Alexandra W.D. Bremers. A computer that sketches along with you. In *Creativity and Cognition*, C&C '22, pages 669–673, New York, NY, USA, June 2022. Association for Computing Machinery. ISBN 978-1-4503-9327-0. doi: 10.1145/3527927.3533732.

[24] J. E. Bresenham. Algorithm for computer control of a digital plotter. *IBM Systems Journal*, 4(1):25–30, 1965. doi: 10.1147/sj.41.0025.

[25] J. C. Briar. *Charts Made Simple: Understanding Knitting Charts Visually : A Knitting on Paper Book*. Glass Iris Publications, 2011. ISBN 978-0-9830792-0-0.

[26] J. C. Briar. Stitch Maps, 2023. URL https://stitch-maps.com/. Accessed 2023.

[27] Amy Bruckman, Kurt Luther, and Casey Fiesler. When should we use real names in published accounts of internet research, 2015.

[28] Leah Buechley and Benjamin Mako Hill. LilyPad in the wild: How hardware's long tail is supporting new engineering and design communities. In *Proceedings of the 8th ACM Conference on Designing Interactive Systems*, DIS '10, pages 199–207, New York, NY, USA, August 2010. Association for Computing Machinery. ISBN 978-1-4503-0103-9. doi: 10.1145/1858171.1858206.

[29] Leah Buechley and Ruby Ta. 3D Printable Play-Dough: New Biodegradable Materials and Creative Possibilities for Digital Fabrication. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, pages 1–15, New York, NY, USA, April 2023. Association for Computing Machinery. ISBN 978-1-4503-9421-5. doi: 10.1145/3544548.3580813.

[30] Leah Buechley, Mike Eisenberg, Jaime Catchen, and Ali Crockett. The LilyPad Arduino: Using computational textiles to investigate engagement, aesthetics, and diversity in computer science education. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '08, pages 423–432, New York, NY, USA, April 2008. Association for Computing Machinery. ISBN 978-1-60558-011-1. doi: 10.1145/1357054.1357123.

[31] Daniel Cardoso Llach. *Builders of the vision : technology and the imagination of design*. Thesis, Massachusetts Institute of Technology, 2012. URL https://dspace.mit.edu/handle/1721.1/77775. Accepted: 2013-03-13T15:45:29Z.

[32] Juan J. Casafranca, Gabriel Cirio, Alejandro Rodríguez, Eder Miguel, and Miguel A. Otaduy. Mixing Yarns and Triangles in Cloth Simulation. *Computer Graphics Forum*, 2020. ISSN 1467-8659. doi: 10.1111/cgf.13915.

[33] Carlos Castillo, Jorge López-Moreno, and Carlos Aliaga. Recent advances in fabric appearance reproduction. *Computers & Graphics*, 84:103–121, 2019. ISSN 0097-8493. doi: https://doi.org/10.1016/j.cag.2019.07.007. URL https://www.sciencedirect.com/science/article/pii/S0097849319301256.

[34] Zekun Chang, Tung D. Ta, Koya Narumi, Heeju Kim, Fuminori Okuya, Dongchi Li, Kunihiro Kato, Jie Qi, Yoshinobu Miyamoto, Kazuya Saito, and Yoshihiro

Kawahara. Kirigami Haptic Swatches: Design Methods for Cut-and-Fold Haptic Feedback Mechanisms. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, pages 1–12, New York, NY, USA, April 2020. Association for Computing Machinery. ISBN 978-1-4503-6708-0. doi: 10.1145/3313831.3376655. URL http://doi.org/10.1145/3313831.3376655.

[35] Amy Cheatle and Steven J. Jackson. Digital Entanglements: Craft, Computation and Collaboration in Fine Art Furniture Production. In *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*, CSCW '15, pages 958–968, New York, NY, USA, February 2015. Association for Computing Machinery. ISBN 978-1-4503-2922-4. doi: 10.1145/2675133.2675291.

[36] Gabriel Cirio, Jorge Lopez-Moreno, and Miguel A. Otaduy. Yarn-Level Cloth Simulation with Sliding Persistent Contacts. *IEEE Transactions on Visualization and Computer Graphics*, 23(2):1152–1162, February 2017. ISSN 1077-2626. doi: 10.1109/TVCG.2016.2592908.

[37] James Coleman, Craig Long, Andrew Manto, and Trygve Wastvedt. Lots of parts, lots of formats, lots of headache. *XRDS: Crossroads, The ACM Magazine for Students*, 22(3):54–57, April 2016. ISSN 1528-4972, 1528-4980. doi: 10.1145/2893501.

[38] K. Compton and M. Mateas. Casual Creators. In *ICCC*, 2015.

[39] Kate Compton and Michael Mateas. A Generative Framework of Generativity. In *Thirteenth Artificial Intelligence and Interactive Digital Entertainment Conference*, September 2017.

[40] Allesandrina Costa. Numbers to GIMP to create images for electronic download, 2018. URL https://alessandrina.com/2018/07/02/numbers-to-gimp-to-create-images-for-electronic-download/.

[41] Mihály Csíkszentmihályi. *Creativity: The Psychology of Discovery and Invention*. HarperCollinsPublishers, 1996. ISBN 978-0-06-017133-9.

[42] DaviWorks. Img2track. https://daviworks.com/knitting/, 2024. https://daviworks.com/knitting/, accessed 2024.

[43] Ashley Del Valle, Mert Toka, Alejandro Aponte, and Jennifer Jacobs. Punchprint: Creating composite fiber-filament craft artifacts by integrating punch needle embroidery and 3d printing. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450394215. doi: 10.1145/3544548.3581298. URL https://doi.org/10.1145/3544548.3581298.

[44] Erik D Demaine, Jason S Ku, and Robert J Lang. A new file standard to represent folded structures. In *Abstr. 26th Fall Workshop Computat. Geometry*, pages 27–28, 2016.

[45] Audrey Desjardins and Timea Tihanyi. ListeningCups: A Case of Data Tactility and Data Stories. In *Proceedings of the 2019 on Designing Interactive Systems Conference*, pages 147–160, San Diego CA USA, June 2019. ACM. ISBN 978-1-4503-5850-7. doi: 10.1145/3322276.3323694.

[46] Laura Devendorf and Kimiko Ryokai. Being the Machine: Reconfiguring Agency and Control in Hybrid Fabrication. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2477–2486, New York, NY, USA, April 2015. Association for Computing Machinery. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702547.

[47] Laura Devendorf, Abigail De Kosnik, Kate Mattingly, and Kimiko Ryokai. Probing the Potential of Post-Anthropocentric 3D Printing. In *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, DIS '16, pages 170–181, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-4031-1. doi: 10.1145/2901790.2901879. URL http://doi.acm.org/10.1145/2901790.2901879.

[48] Laura Devendorf, Katya Arquilla, Sandra Wirtanen, Allison Anderson, and Steven Frost. Craftspeople as Technical Collaborators: Lessons Learned through an Experimental Weaving Residency. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, pages 1–13, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 978-1-4503-6708-0. doi: 10.1145/3313831.3376820. URL https://doi.org/10.1145/3313831.3376820. event-place: Honolulu, HI, USA.

[49] Laura Devendorf, Kathryn Walters, Marianne Fairbanks, Etta Sandry, and Emma R Goodwill. AdaCAD: Parametric Design as a New Form of Notation for Complex Weaving. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, pages 1–18, New York, NY, USA, April 2023. Association for Computing Machinery. ISBN 978-1-4503-9421-5. doi: 10.1145/3544548.3581571.

[50] Kristin N. Dew, Samantha Shorey, and Daniela Rosner. Making within limits: Towards salvage fabrication. In *Proceedings of the 2018 Workshop on Computing within Limits*, LIMITS '18, pages 1–11, New York, NY, USA, May 2018. Association for Computing Machinery. ISBN 978-1-4503-6575-8. doi: 10.1145/3232617.3232626.

[51] Paul Dourish. User experience as legitimacy trap. *Interactions*, 26(6):46–49, 2019.

[52] B. Eberhardt and A. Weber. A particle system approach to knitted textiles. *Computers & Graphics*, 23(4):599–606, 1999. ISSN 0097-8493. doi: https://doi.org/10.1016/S0097-8493(99)00077-1. URL https://www.sciencedirect.com/science/article/pii/S0097849399000771.

[53] Tamara Anna Efrat, Moran Mizrahi, and Amit Zoran. The Hybrid Bricolage: Bridging Parametric Design with Craft through Algorithmic Modularity. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 5984–5995, New York, NY, USA, May 2016. Association for Computing Machinery. ISBN 978-1-4503-3362-7. doi: 10.1145/2858036.2858441. URL http://doi.org/10.1145/2858036.2858441.

[54] Madeleine Clare Elish. The stakes of uncertainty: Developing and integrating machine learning in clinical care. In *Ethnographic Praxis in Industry Conference Proceedings*, pages 364–380. Wiley Online Library, 2018.

[55] Shreyosi Endow, Mohammad Abu Nasir Rakib, Anvay Srivastava, Sara Rastegarpouyani, and Cesar Torres. Embr: A Creative Framework for Hand Embroidered Liquid Crystal Textile Displays. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, pages 1–14, New York, NY,

USA, April 2022. Association for Computing Machinery. ISBN 978-1-4503-9157-3. doi: 10.1145/3491102.3502117.

[56] Sarah Evans, Katie Davis, Abigail Evans, Julie Ann Campbell, David P. Randall, Kodlee Yin, and Cecilia Aragon. More Than Peer Production: Fanfiction Communities as Sites of Distributed Mentoring. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, CSCW '17, pages 259–272, New York, NY, USA, February 2017. Association for Computing Machinery. ISBN 978-1-4503-4335-0. doi: 10.1145/2998181.2998342. URL http://doi.org/10.1145/2998181.2998342.

[57] Boris Ewenstein and Jennifer Whyte. Knowledge Practices in Design: The Role of Visual Representations as 'Epistemic Objects'. *Organization Studies*, 30(1): 07–30, January 2009. ISSN 0170-8406. doi: 10.1177/0170840608083014. URL https://doi.org/10.1177/0170840608083014.

[58] Stitch Fiddle. Stitch Fiddle | Online knitting and cross stitch chart pattern editor software, 2023. URL https://www.stitchfiddle.com/. Accessed 2023.

[59] Gerhard Fischer, Kumiyo Nakakoji, and Yunwen Ye. Metadesign: Guidelines for Supporting Domain Experts in Software Development. *IEEE Software*, 26(5): 37–44, September 2009. ISSN 1937-4194. doi: 10.1109/MS.2009.134.

[60] Jack Forman, Mustafa Doga Dogan, Hamilton Forsythe, and Hiroshi Ishii. Defextiles: 3d printing quasi-woven fabric via under-extrusion. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, page 1222–1233, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375146. doi: 10.1145/3379337.3415876. URL https://doi.org/10.1145/3379337.3415876.

[61] Jack Forman, Ozgun Kilic Afsar, Sarah Nicita, Rosalie Hsin-Ju Lin, Liu Yang, Megan Hofmann, Akshay Kothakonda, Zachary Gordon, Cedric Honnet, Kristen Dorsey, Neil Gershenfeld, and Hiroshi Ishii. FibeRobo: Fabricating 4D Fiber Interfaces by Continuous Drawing of Temperature Tunable Liquid Crystal Elastomers. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, pages 1–17, New York, NY, USA, Octo-

ber 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606732.

[62] Frikk Fossdal, Rogardt Heldal, and Nadya Peek. Interactive Digital Fabrication Machine Control Directly Within a CAD Environment. In *Symposium on Computational Fabrication*, SCF '21, pages 1–15, New York, NY, USA, October 2021. Association for Computing Machinery. ISBN 978-1-4503-9090-3. doi: 10.1145/3485114.3485120.

[63] Frikk H. Fossdal, Jens Dyvik, Jakob Anders Nilsson, Jon Nordby, Torbjørn Nordvik Helgesen, Rogardt Heldal, and Nadya Peek. Fabricatable Machines: A Toolkit for Building Digital Fabrication Machines. In *Proceedings of the Fourteenth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '20, pages 411–422, Sydney NSW, Australia, February 2020. Association for Computing Machinery. ISBN 978-1-4503-6107-1. doi: 10.1145/3374920.3374929.

[64] Frikk H Fossdal, Vinh Nguyen, Rogardt Heldal, Corie L. Cobb, and Nadya Peek. Vespidae: A programming framework for developing digital fabrication workflows. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*, DIS '23, page 2034–2049, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450398930. doi: 10.1145/3563657.3596106. URL https://doi.org/10.1145/3563657.3596106.

[65] Raune Frankjær and Peter Dalsgaard. Understanding Craft-Based Inquiry in HCI. In *Proceedings of the 2018 Designing Interactive Systems Conference*, DIS '18, pages 473–484, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 978-1-4503-5198-0. doi: 10.1145/3196709.3196750. URL https://doi.org/10.1145/3196709.3196750. event-place: Hong Kong, China.

[66] Mikhaila Friske, Shanel Wu, and Laura Devendorf. AdaCAD: Crafting Software For Smart Textiles Design. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1–13, New York, NY, USA, May 2019. Association for Computing Machinery. ISBN 978-1-4503-5970-2.

[67] Clifford Geertz. *The interpretation of cultures: selected essays*. Basic Books, New York, 1973. ISBN 978-0-465-09719-7 978-0-465-03425-3.

[68] Neil Gershenfeld. How to make almost anything, Dec 2015. ISSN 0015-7120. URL https://www.foreignaffairs.com/articles/2012-09-27/how-make-almost-anything.

[69] Timothy Gerstner, Doug DeCarlo, Marc Alexa, Adam Finkelstein, Yotam Gingold, and Andrew Nealen. Pixelated Image Abstraction. In Paul Asente and Cindy Grimm, editors, *International Symposium on Non-Photorealistic Animation and Rendering*. The Eurographics Association, 2012. ISBN 978-3-905673-90-6. doi: 10.2312/PE/NPAR/NPAR12/029-036.

[70] Maas Goudswaard, Abel Abraham, Bruna Goveia da Rocha, Kristina Andersen, and Rong-Hao Liang. FabriClick: Interweaving Pushbuttons into Fabrics Using 3D Printing and Digital Embroidery. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, pages 379–393, New York, NY, USA, July 2020. Association for Computing Machinery. ISBN 978-1-4503-6974-9. doi: 10.1145/3357236.3395569.

[71] Bruna Goveia Da Rocha, Oscar Tomico, Panos Markopoulos, and Daniel Tetteroo. Crafting Research Products through Digital Machine Embroidery. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, pages 341–350, Eindhoven Netherlands, July 2020. ACM. ISBN 978-1-4503-6974-9. doi: 10.1145/3357236.3395443.

[72] Bruna Goveia da Rocha, Johannes M. L. van der Kolk, and Kristina Andersen. Exquisite Fabrication: Exploring Turn-taking between Designers and Digital Fabrication Machines. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pages 1–9, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8096-6. doi: 10.1145/3411764.3445236.

[73] Bruna Goveia da Rocha, Kristina Andersen, and Oscar Tomico. Portfolio of Loose Ends. In *Designing Interactive Systems Conference*, pages 527–540, Virtual Event Australia, June 2022. ACM. ISBN 978-1-4503-9358-4. doi: 10.1145/3532106.3533516.

[74] Mary L. Gray and Siddharth Suri. *Ghost Work: How to Stop Silicon Valley from Building a New Global Underclass*. HMH Books, 2019. ISBN 9781328566287.

[75] Saul Greenberg. Toolkits and interface creativity. *Multimedia Tools and Applications*, 32(2):139–159, February 2007. ISSN 1573-7721. doi: 10.1007/s11042-006-0062-y.

[76] Susan Guagliumi. *Hand-Manipulated Stitches for Machine Knitters*. GUAGLIUMIDOTCOM, August 1990. ISBN 978-1-73331-211-0.

[77] Susan Guagliumi. *Open Spaces: Machine Knit Eyelets, Ladders and Slits*. Guagliumidotcom, April 2018. ISBN 978-1-73331-217-2.

[78] Nur Al-huda Hamdan, Simon Voelker, and Jan Borchers. Sketch&Stitch: Interactive Embroidery for E-textiles. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–13, New York, NY, USA, April 2018. Association for Computing Machinery. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3173656.

[79] Richard R Hamming. *The Art of Doing Science and Engineering: Learning to Learn*. CRC Press, 1997.

[80] Richard Wesley Hamming. *Numerical Methods for Scientists and Engineers*. McGraw-Hill, 1973. ISBN 978-0-07-025887-7.

[81] Shiqing He and Eytan Adar. Plotting with Thread: Fabricating Delicate Punch Needle Embroidery with X-Y Plotters. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, pages 1047–1057, New York, NY, USA, July 2020. Association for Computing Machinery. ISBN 978-1-4503-6974-9. doi: 10.1145/3357236.3395540.

[82] Donald Hearn, Pauline Baker, and Warren Carithers. *Computer Graphics with Open GL*. Pearson, fourth edition, 2010.

[83] Kathryn Henderson. Flexible Sketches and Inflexible Data Bases: Visual Communication, Conscription Devices, and Boundary Objects in Design Engineering. *Science, Technology, & Human Values*, 16(4):448–473, October 1991. ISSN 0162-2439. doi: 10.1177/016224399101600402. URL https://doi.org/10.1177/016224399101600402.

[84] Benjamin Mako Hill and Andrés Monroy-Hernández. The cost of collaboration for code and art: evidence from a remixing community. In *Proceedings of the 2013 conference on Computer supported cooperative work*, CSCW '13, pages 1035–1046, New York, NY, USA, February 2013. Association for Computing Machinery. ISBN 978-1-4503-1331-5. doi: 10.1145/2441776.2441893. URL http://doi.org/10.1145/2441776.2441893.

[85] Mare Hirsch, Gabrielle Benabdallah, Jennifer Jacobs, and Nadya Peek. Nothing like compilation: How professional digital fabrication workflows go beyond extruding, milling, and machines. *ACM Trans. Comput.-Hum. Interact.*, jul 2023. ISSN 1073-0516. doi: 10.1145/3609328. URL https://doi.org/10.1145/3609328. Just Accepted.

[86] Megan Hofmann, Gabriella Hann, Scott E. Hudson, and Jennifer Mankoff. 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*, CHI '18, page 1–12, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356206. doi: 10.1145/3173574.3173875. URL https://doi.org/10.1145/3173574.3173875.

[87] Megan Hofmann, Lea Albaugh, Ticha Sethapakadi, Jessica Hodgins, Scott E. Hudson, James McCann, and Jennifer Mankoff. 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, pages 5–16, New York, NY, USA, October 2019. Association for Computing Machinery. ISBN 978-1-4503-6816-2. doi: 10.1145/3332165.3347886.

[88] Megan Hofmann, Jennifer Mankoff, and Scott E. Hudson. KnitGIST: A Programming Synthesis Toolkit for Generating Functional Machine-Knitting Textures. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, pages 1234–1247, New York, NY, USA, October 2020. Association for Computing Machinery. ISBN 978-1-4503-7514-6. doi: 10.1145/3379337.3415590.

[89] Megan Hofmann, Lea Albaugh, Tongyan Wang, Jennifer Mankoff, and Scott E

Hudson. KnitScript: A Domain-Specific Scripting Language for Advanced Machine Knitting. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, pages 1–21, New York, NY, USA, October 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606789.

[90] Nathaniel Hudson, Celena Alcock, and Parmit K. Chilana. Understanding Newcomers to 3D Printing: Motivations, Workflows, and Barriers of Casual Makers. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, pages 384–396, New York, NY, USA, May 2016. Association for Computing Machinery. ISBN 978-1-4503-3362-7. doi: 10.1145/2858036.2858266. URL http://doi.org/10.1145/2858036.2858266.

[91] Scott E. Hudson. Printing teddy bears: A technique for 3D printing of soft interactive objects. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '14, pages 459–468, New York, NY, USA, April 2014. Association for Computing Machinery. ISBN 978-1-4503-2473-1. doi: 10.1145/2556288.2557338.

[92] Scott E. Hudson and Jennifer Mankoff. Concepts, Values, and Methods for Technical Human–Computer Interaction Research. In Judith S. Olson and Wendy A. Kellogg, editors, *Ways of Knowing in HCI*, pages 69–93. Springer New York, New York, NY, 2014. ISBN 978-1-4939-0377-1 978-1-4939-0378-8. doi: 10.1007/978-1-4939-0378-84.

[93] Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. Knitty: 3D Modeling of Knitted Animals with a Production Assistant Interface. In *Eurographics (Short Papers)*. The Eurographics Association, 2008. doi: 10.2312/egs.20081011.

[94] Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. Knitting a 3d model. *Computer Graphics Forum*, 27(7):1737–1743, 2008. doi: https://doi.org/10.1111/j.1467-8659.2008.01318.x. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/j.1467-8659.2008.01318.x.

[95] Cochenille Computer Knit Products Inc. Cochenille Design Studio, 2016. Accessed 2023.

[96] Tiffany C. Inglis and Craig S. Kaplan. Pixelating vector line art. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '12, page 21–28, Goslar, DEU, 2012. Eurographics Association. ISBN 9783905673906.

[97] Tiffany C. Inglis, Daniel Vogel, and Craig S. Kaplan. Rasterizing and antialiasing vector line art in the pixel art style. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering*, NPAR '13, page 25–32, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450321983. doi: 10.1145/2486042.2486044. URL https://doi.org/10.1145/2486042.2486044.

[98] Tim Ingold. *Making: anthropology, archaeology, art and architecture*. Routledge, London ; New York, 2013. ISBN 978-0-415-56722-0 978-0-415-56723-7.

[99] Sarah Inman and David Ribes. "Beautiful Seams": Strategic Revelations and Concealments. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pages 1–14, Glasgow Scotland Uk, May 2019. ACM. ISBN 978-1-4503-5970-2. doi: 10.1145/3290605.3300508.

[100] Alexandra Ion, Johannes Frohnhofen, Ludwig Wall, Robert Kovacs, Mirela Alistar, Jack Lindsay, Pedro Lopes, Hsiang-Ting Chen, and Patrick Baudisch. Metamaterial Mechanisms. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 529–539, New York, NY, USA, October 2016. Association for Computing Machinery. ISBN 978-1-4503-4189-9. doi: 10.1145/2984511.2984540.

[101] Lilly Irani. *Chasing Innovation: Making Entrepreneurial Citizens in Modern India*. Princeton Studies in Culture and Technology. Princeton University Press, 2019. ISBN 9780691189444.

[102] Jennifer Jacobs and Leah Buechley. Codeable objects: Computational design and digital fabrication for novice programmers. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 1589–1598, Paris France, April 2013. ACM. ISBN 978-1-4503-1899-0. doi: 10.1145/2470654.2466211.

[103] Jennifer Jacobs and Amit Zoran. Hybrid Practice in the Kalahari: Design Collaboration through Digital Tools and Hunter-Gatherer Craft. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 619–628, New York, NY, USA, 2015. Association for Computing Machinery. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702362. URL https://doi.org/10.1145/2702123.2702362. event-place: Seoul, Republic of Korea.

[104] Jennifer Jacobs, David Mellis, Amit Zoran, Cesar Torres, Joel Brandt, and Theresa Jean Tanenbaum. Digital Craftsmanship: HCI Takes on Technology as an Expressive Medium. In *Proceedings of the 2016 ACM Conference Companion Publication on Designing Interactive Systems*, DIS '16 Companion, pages 57–60, New York, NY, USA, June 2016. Association for Computing Machinery. ISBN 978-1-4503-4315-2. doi: 10.1145/2908805.2913018. URL http://doi.org/10.1145/2908805.2913018.

[105] Jennifer Jacobs, Joel Brandt, Radomír Mech, and Mitchel Resnick. Extending Manual Drawing Practices with Artist-Centric Programming Tools. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–13, New York, NY, USA, April 2018. Association for Computing Machinery. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3174164. URL http://doi.org/10.1145/3173574.3174164.

[106] Benjamin Jones, Yuxuan Mei, Haisen Zhao, Taylor Gotfrid, Jennifer Mankoff, and Adriana Schulz. Computational Design of Knit Templates. *ACM Transactions on Graphics*, 41(2):1–16, April 2022. ISSN 0730-0301, 1557-7368. doi: 10.1145/3488006.

[107] Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Simulating knitted cloth at the yarn level. In *ACM SIGGRAPH 2008 Papers*, SIGGRAPH '08, New York, NY, USA, 2008. Association for Computing Machinery. ISBN 9781450301121. doi: 10.1145/1399504.1360664. URL https://doi.org/10.1145/1399504.1360664.

[108] Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Efficient yarn-based cloth with adaptive contact linearization. *ACM Trans. Graph.*, 29(4), jul 2010.

ISSN 0730-0301. doi: 10.1145/1778765.1778842. URL https://doi.org/10.1145/1778765.1778842.

[109] Levi Kapllani, Chelsea Amanatides, Genevieve Dion, Vadim Shapiro, and David E. Breen. Topoknit: A process-oriented representation for modeling the topology of yarns in weft-knitted textiles. *Graphical Models*, 118:101114, 2021. ISSN 1524-0703. doi: https://doi.org/10.1016/j.gmod.2021.101114. URL https://www.sciencedirect.com/science/article/pii/S1524070321000199.

[110] Levi Kapllani, Chelsea Amanatides, Genevieve Dion, and David E. Breen. Loop Order Analysis of Weft-Knitted Textiles. *Textiles*, 2(2):275–295, June 2022. ISSN 2673-7248. doi: 10.3390/textiles2020015.

[111] Ayelet Karmon, Yoav Sterman, Tom Shaked, Eyal Sheffer, and Shoval Nir. KNITIT: A computational tool for design, simulation, and fabrication of multiple structured knits. In *Proceedings of the 2nd ACM Symposium on Computational Fabrication*, pages 1–10, Cambridge Massachusetts, June 2018. ACM. ISBN 978-1-4503-5854-5. doi: 10.1145/3213512.3213516.

[112] Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. 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, pages 53–65, New York, NY, USA, October 2019. Association for Computing Machinery. ISBN 978-1-4503-6816-2. doi: 10.1145/3332165.3347879.

[113] Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. 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, page 53–65, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450368162. doi: 10.1145/3332165.3347879. URL https://doi.org/10.1145/3332165.3347879.

[114] Alexandre Kaspar, Tae-Hyun Oh, Liane Makatura, Petr Kellnhofer, and Wojciech Matusik. Neural inverse knitting: From images to manufacturing instructions. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the*

36th International Conference on Machine Learning, volume 97 of *Proceedings of Machine Learning Research*, pages 3272–3281, Long Beach, California, USA, 09–15 Jun 2019. PMLR. URL http://proceedings.mlr.press/v97/kaspar19a.html.

[115] Alexandre Kaspar, Kui Wu, Yiyue Luo, Liane Makatura, and Wojciech Matusik. Knit sketching: From cut & sew patterns to machine-knit garments. *ACM Transactions on Graphics*, 40(4):1–15, August 2021. ISSN 0730-0301, 1557-7368. doi: 10.1145/3450626.3459752.

[116] Jeeeun Kim, Haruki Takahashi, Homei Miyashita, Michelle Annett, and Tom Yeh. Machines as Co-Designers: A Fiction on the Future of Human-Fabrication Machine Interaction. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, pages 790–805, New York, NY, USA, May 2017. Association for Computing Machinery. ISBN 978-1-4503-4656-6. doi: 10.1145/3027063.3052763.

[117] Jeeeun Kim, Clement Zheng, Haruki Takahashi, Mark D Gross, Daniel Ashbrook, and Tom Yeh. Compositional 3D printing: Expanding & supporting workflows towards continuous fabrication. In *Proceedings of the 2nd ACM Symposium on Computational Fabrication*, SCF '18, pages 1–10, New York, NY, USA, June 2018. Association for Computing Machinery. ISBN 978-1-4503-5854-5. doi: 10.1145/3213512.3213518.

[118] Jin Hee (Heather) Kim, Kunpeng Huang, Simone White, Melissa Conroy, and Cindy Hsin-Liu Kao. KnitDermis: Fabricating Tactile On-Body Interfaces Through Machine Knitting. In *Proceedings of the 2021 ACM Designing Interactive Systems Conference*, DIS '21, pages 1183–1200, New York, NY, USA, June 2021. Association for Computing Machinery. ISBN 978-1-4503-8476-6. doi: 10.1145/3461778.3462007.

[119] Joy Kim, Maneesh Agrawala, and Michael S. Bernstein. Mosaic: Designing Online Creative Communities for Sharing Works-in-Progress. In *Proceedings of the 2017 ACM Conference on Computer Supported Cooperative Work and Social Computing*, pages 246–258, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-

4335-0. doi: 10.1145/2998181.2998195. URL http://doi.acm.org/10.1145/2998181.2998195.

[120] Bill King. *Machine Knitting Techniques: Cables*. Crowood Press, February 2023. ISBN 978-0-7198-4193-4.

[121] Karin D Knorr. Tinkering toward success: Prelude to a theory of scientific practice'. *Theory and Society*, 8:347–76, 1979.

[122] Marion Koelle, Madalina Nicolae, Aditya Shekhar Nittala, Marc Teyssier, and Jürgen Steimle. Prototyping soft devices with interactive bioplastics. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393201. doi: 10.1145/3526113.3545623. URL https://doi.org/10.1145/3526113.3545623.

[123] Robert Kovacs, Anna Seufert, Ludwig Wall, Hsiang-Ting Chen, Florian Meinel, Willi Müller, Sijing You, Maximilian Brehm, Jonathan Striebel, Yannis Kommana, Alexander Popiak, Thomas Bläsius, and Patrick Baudisch. TrussFab: Fabricating Sturdy Large-Scale Structures on Desktop 3D Printers. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 2606–2616, New York, NY, USA, May 2017. Association for Computing Machinery. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3026016. URL http://doi.org/10.1145/3025453.3026016.

[124] Stacey Kuznetsov and Eric Paulos. Rise of the expert amateur: DIY projects, communities, and cultures. In *Proceedings of the 6th Nordic Conference on Human-Computer Interaction: Extending Boundaries*, NordiCHI '10, pages 295–304, New York, NY, USA, October 2010. Association for Computing Machinery. ISBN 978-1-60558-934-3. doi: 10.1145/1868914.1868950. URL http://doi.org/10.1145/1868914.1868950.

[125] Sophie Landwehr Sydow, Martin Jonsson, and Jakob Tholander. Modding the Pliable Machine: Unpacking the Creative and Social Practice of Upkeep at the Makerspace. In *Creativity and Cognition*, C&C '22, pages 220–233, New York, NY, USA, June 2022. Association for Computing Machinery. ISBN 978-1-4503-9327-0. doi: 10.1145/3527927.3532804.

[126] Bruno Latour and Steve Woolgar. *Laboratory Life: The Construction of Scientific Facts*. Princeton University Press, 1979.

[127] Eldy S. Lazaro Vasquez, Netta Ofer, Shanel Wu, Mary Etta West, Mirela Alistar, and Laura Devendorf. Exploring biofoam as a material for tangible interaction. In *Proceedings of the 2022 ACM Designing Interactive Systems Conference*, DIS '22, page 1525–1539, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393584. doi: 10.1145/3532106.3533494. URL https://doi.org/10.1145/3532106.3533494.

[128] G. A. V. Leaf and A. Glaskin. The geometry of a plain knitted loop. *Journal of the Textile Institute Transactions*, 46(9):T587–T605, 1955. doi: 10.1080/19447027.1955.10750345.

[129] Jonathan Leaf, Rundong Wu, Eston Schweickart, Doug L. James, and Steve Marschner. Interactive design of periodic yarn-level cloth patterns. *ACM Trans. Graph.*, 37(6), dec 2018. ISSN 0730-0301. doi: 10.1145/3272127.3275105. URL https://doi.org/10.1145/3272127.3275105.

[130] Mackenzie Leake, Frances Lai, Tovi Grossman, Daniel Wigdor, and Ben Lafreniere. PatchProv: Supporting Improvisational Design Practices for Modern Quilting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pages 1–17, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8096-6. doi: 10.1145/3411764.3445601.

[131] Mackenzie Leake, Gilbert Bernstein, and Maneesh Agrawala. Sketch-Based Design of Foundation Paper Pieceable Quilts. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, pages 1–11, New York, NY, USA, October 2022. Association for Computing Machinery. ISBN 978-1-4503-9320-1. doi: 10.1145/3526113.3545643.

[132] Mackenzie Leake, Kathryn Jin, Abe Davis, and Stefanie Mueller. InStitches: Augmenting Sewing Patterns with Personalized Material-Efficient Practice. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, pages 1–14, Hamburg Germany, April 2023. ACM. ISBN 978-1-4503-9421-5. doi: 10.1145/3544548.3581499.

[133] David Ledo, Steven Houben, Jo Vermeulen, Nicolai Marquardt, Lora Oehlberg, and Saul Greenberg. Evaluation Strategies for HCI Toolkit Research. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–17, New York, NY, USA, April 2018. Association for Computing Machinery. ISBN 978-1-4503-5620-6.

[134] Yi-Chin Lee and Lea Albaugh. Hybrid embroidery games: Playing with materials, machines, and people. In *Proceedings of the 2021 ACM Designing Interactive Systems Conference*, DIS '21, page 749–762, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384766. doi: 10.1145/3461778.3462019. URL https://doi.org/10.1145/3461778.3462019.

[135] Danny Leen, Tom Veuskens, Kris Luyten, and Raf Ramakers. Jigfab: Computational fabrication of constraints to facilitate woodworking with power tools. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, page 1–12, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450359702. doi: 10.1145/3290605.3300386. URL https://doi.org/10.1145/3290605.3300386.

[136] Danny Leen, Nadya Peek, and Raf Ramakers. Lamifold: Fabricating objects with integrated mechanisms using a laser cutter lamination workflow. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology*, UIST '20, page 304–316, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450375146. doi: 10.1145/3379337.3415885. URL https://doi.org/10.1145/3379337.3415885.

[137] Jingyi Li, Sonia Hashim, and Jennifer Jacobs. What we can learn from visual artists about software development. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450380966. doi: 10.1145/3411764.3445682. URL https://doi.org/10.1145/3411764.3445682.

[138] Jingyi Li, Eric Rawn, Jacob Ritchie, Jasper Tran O'Leary, and Sean Follmer. Beyond the Artifact: Power as a Lens for Creativity Support Tools. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technol-*

ogy, UIST '23, pages 1–15, New York, NY, USA, October 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606831.

[139] Jenny Lin, Vidya Narayanan, and James McCann. Efficient transfer planning for flat knitting. In *Proceedings of the 2nd ACM Symposium on Computational Fabrication*, SCF '18, pages 1–7, New York, NY, USA, June 2018. Association for Computing Machinery. ISBN 978-1-4503-5854-5. doi: 10.1145/3213512.3213515.

[140] Jenny Lin, Vidya Narayanan, Yuka Ikarashi, Jonathan Ragan-Kelley, Gilbert Bernstein, and James McCann. Semantics and Scheduling for Machine Knitting Compilers. *ACM Transactions on Graphics*, 42(4):143:1–143:26, July 2023. ISSN 0730-0301. doi: 10.1145/3592449.

[141] Jenny Lin, Vidya Narayanan, Yuka Ikarashi, Jonathan Ragan-Kelley, Gilbert Bernstein, and James McCann. Semantics and Scheduling for Machine Knitting Compilers. *ACM Transactions on Graphics*, 42(4):1–26, August 2023. ISSN 0730-0301, 1557-7368. doi: 10.1145/3592449.

[142] Silvia M. Lindtner. *Prototype Nation: China and the Contested Promise of Innovation*. Princeton Studies in Culture and Technology. Princeton University Press, 2020. ISBN 9780691204956.

[143] Jen Liu, Daragh Byrne, and Laura Devendorf. Design for Collaborative Survival: An Inquiry into Human-Fungi Relationships. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–13, New York, NY, USA, April 2018. Association for Computing Machinery. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3173614.

[144] Zishun Liu, Xingjian Han, Yuchen Zhang, Xiangjia Chen, Yu-Kun Lai, Eugeni L. Doubrovski, Emily Whiting, and Charlie C. L. Wang. Knitting 4D garments with elasticity controlled for body motion. *ACM Transactions on Graphics*, 40(4):62:1–62:16, July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459868.

[145] Jesse Louis-Rosenberg. Drowning in triangle soup: the quest for a better 3-D printing file format. *XRDS: Crossroads, The ACM Magazine for Students*, 22 (3):58–62, April 2016. ISSN 1528-4972. doi: 10.1145/2893503. URL http://doi.org/10.1145/2893503.

[146] Soft Byte Ltd. DesignaKnit 9 - Knitwear design software., 2021. URL https://www.softbyte.co.uk/designaknit9.htm. Accessed 2023.

[147] Yiyue Luo, Kui Wu, Tomás Palacios, and Wojciech Matusik. KnitUI: Fabricating Interactive and Sensing Textiles with Machine Knitting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, number 668 in CHI '21, pages 1–12. Association for Computing Machinery, New York, NY, USA, May 2021. ISBN 978-1-4503-8096-6.

[148] Yiyue Luo, Kui Wu, Andrew Spielberg, Michael Foshey, Daniela Rus, Tomás Palacios, and Wojciech Matusik. Digital Fabrication of Pneumatic Actuators with Integrated Sensing by Machine Knitting. In *CHI Conference on Human Factors in Computing Systems*, pages 1–13, New Orleans LA USA, April 2022. ACM. ISBN 978-1-4503-9157-3. doi: 10.1145/3491102.3517577.

[149] Li-Ke Ma, Yizhonc Zhang, Yang Liu, Kun Zhou, and Xin Tong. Computational design and fabrication of soft pneumatic objects with desired deformations. *ACM Transactions on Graphics*, 36(6):239:1–239:12, November 2017. ISSN 0730-0301. doi: 10.1145/3130800.3130850. URL http://doi.org/10.1145/3130800.3130850.

[150] Anne L. MacDonald. *No Idle Hands: The Social History of American Knitting*. Random House Publishing Group, November 2010. ISBN 978-0-307-77544-3.

[151] James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. A compiler for 3d machine knitting. *ACM Trans. Graph.*, 35(4), jul 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925940. URL https://doi.org/10.1145/2897824.2925940.

[152] James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. A compiler for 3D machine knitting. *ACM Transactions on Graphics*, 35(4):49:1–49:11, July 2016. ISSN 0730-0301. doi: 10.1145/2897824.2925940.

[153] Malcolm McCullough. *Abstracting Craft: The Practiced Digital Hand*. MIT Press, 1998. ISBN 978-0-262-63189-1.

[154] Denisa Qori McDonald, Richard Vallett, Erin Solovey, Geneviève Dion, and Ali Shokoufandeh. Knitted Sensors: Designs and Novel Approaches for Real-Time, Real-World Sensing. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, 4(4):145:1–145:25, December 2020. doi: 10.1145/3432201.

[155] A. M. Mehta and D. Rus. An end-to-end system for designing mechanical structures for print-and-fold robots. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1460–1465, New York, NY, May 2014. Institute of Electrical and Electronics Engineers (IEEE). doi: 10.1109/ICRA.2014.6907044.

[156] M. Meißner and B. Eberhardt. The art of knitted fabrics, realistic & physically based modelling of knitted patterns. *Computer Graphics Forum*, 17(3): 355–362, 1998. doi: https://doi.org/10.1111/1467-8659.00282. URL https://onlinelibrary.wiley.com/doi/abs/10.1111/1467-8659.00282.

[157] Michael Hansmeyer. Subdivided Columns, 2010.

[158] Chart Minder. Chart Minder, 2023. URL https://www.chart-minder.com/. Accessed 2023.

[159] Rahul Mitra, Liane Makatura, Emily Whiting, and Edward Chien. Helix-free stripes for knit graph design. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701597. doi: 10.1145/3588432.3591564. URL https://doi.org/10.1145/3588432.3591564.

[160] Mako Miyatake, Koya Narumi, Yuji Sekiya, and Yoshihiro Kawahara. Flower Jelly Printer: Slit Injection Printing for Parametrically Designed Flower Jelly. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pages 1–10, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8096-6. doi: 10.1145/3411764.3445346.

[161] Annemarie Mol. *The Body Multiple: Ontology in Medical Practice*. Science and Cultural Theory. Duke University Press, 2002. ISBN 9780822384151.

[162] Andrés Monroy-Hernández, Benjamin Mako Hill, Jazmin Gonzalez-Rivero, and danah boyd. Computers can't give credit: how automatic attribution falls short in an online remixing community. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '11, pages 3421–3430, New York, NY, USA, May 2011. Association for Computing Machinery. ISBN 978-1-4503-0228-9. doi: 10.1145/1978942.1979452. URL http://doi.org/10.1145/1978942.1979452.

[163] Anders Mørch. Three levels of end-user tailoring: Customization, integration, and extension. In *Computers and Design in Context*, pages 51–76. MIT Press, Cambridge, MA, USA, November 1997. ISBN 978-0-262-11223-9.

[164] Yuki Mori and Takeo Igarashi. Plushie: An interactive design system for plush toys. In *ACM SIGGRAPH 2007 Papers*, SIGGRAPH '07, pages 45–es, New York, NY, USA, July 2007. Association for Computing Machinery. ISBN 978-1-4503-7836-9. doi: 10.1145/1275808.1276433.

[165] Stefanie Mueller, Pedro Lopes, and Patrick Baudisch. Interactive construction: Interactive fabrication of functional mechanical devices. In *Proceedings of the 25th Annual ACM Symposium on User Interface Software and Technology*, pages 599–606, New York, NY, USA, October 2012. Association for Computing Machinery. ISBN 978-1-4503-1580-7.

[166] Stefanie Mueller, Sangha Im, Serafima Gurevich, Alexander Teibrich, Lisa Pfisterer, François Guimbretière, and Patrick Baudisch. WirePrint: 3D printed previews for fast prototyping. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*, UIST '14, pages 273–280, New York, NY, USA, October 2014. Association for Computing Machinery. ISBN 978-1-4503-3069-5. doi: 10.1145/2642918.2647359.

[167] D. L. Munden. The geometry and dimensional properties of plain-knit fabrics. *Journal of the Textile Institute Transactions*, 50(7):T448–T471, 1959. doi: 10.1080/19447025908659923.

[168] Brad Myers, Scott E. Hudson, and Randy Pausch. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction*, 7(1):3–28, March 2000. ISSN 1073-0516. doi: 10.1145/344949.344959.

[169] Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James Mccann. Automatic Machine Knitting of 3D Meshes. *ACM Transactions on Graphics*, 37(3):35:1–35:15, August 2018. ISSN 0730-0301. doi: 10.1145/3186265.

[170] Vidya Narayanan, Kui Wu, Cem Yuksel, and James McCann. Visual knitting machine programming. *ACM Transactions on Graphics*, 38(4):63:1–63:13, July 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322995.

[171] Vidya Narayanan*, Kui Wu*, Cem Yuksel, and Jim McCann. Visual knitting machine programming. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2019)*, 38(4):63:1–63:13, jul 2019. ISSN 0730-0301. doi: 10.1145/3306346.3322995. URL http://doi.acm.org/10.1145/3306346.3322995. (*Joint First Authors).

[172] Bonnie A. Nardi. *A Small Matter of Programming: Perspectives on End User Computing*. MIT Press, Cambridge, MA, 1993. ISBN 978-0-262-14053-9.

[173] Bettina Nissen and John Bowers. Data-Things: Digital Fabrication Situated within Participatory Data Translation Activities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 2467–2476, New York, NY, USA, April 2015. Association for Computing Machinery. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702245.

[174] David F. Noble. *Forces of production: a social history of industrial automation*. Knopf, New York, 1st ed edition, 1984. ISBN 978-0-394-51262-4.

[175] Behnaz Norouzi, Marianne Kinnula, and Netta Iivari. Making Sense of 3D Modelling and 3D Printing Activities of Young People: A Nexus Analytic Inquiry. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pages 1–16, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8096-6. doi: 10.1145/3411764.3445139.

[176] Lora Oehlberg, Wesley Willett, and Wendy E. Mackay. Patterns of Physical Design Remixing in Online Maker Communities. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 639–648, New York, NY, USA, April 2015. Association for Computing Machinery. ISBN 978-1-4503-3145-6. doi: 10.1145/2702123.2702175. URL http://doi.org/10.1145/2702123.2702175.

[177] Hyunjoo Oh, Jeeeun Kim, Cory Morales, Mark Gross, Michael Eisenberg, and Sherry Hsi. FoldMecha: Exploratory Design and Engineering of Mechanical Papercraft. In *Proceedings of the Eleventh International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '17, pages 131–139, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 978-1-4503-4676-4. doi: 10.1145/3024969.3024991. URL https://doi.org/10.1145/3024969.3024991. event-place: Yokohama, Japan.

[178] Dan R. Olsen. Evaluating user interface systems research. In *Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology*, UIST '07, pages 251–258, New York, NY, USA, October 2007. Association for Computing Machinery. ISBN 978-1-59593-679-0. doi: 10.1145/1294211.1294256.

[179] openFrameworks. openFrameworks Forum, 2020. URL https://forum.openframeworks.cc/.

[180] Jifei Ou, Daniel Oran, Don Derek Haddad, Joseph Paradiso, and Hiroshi Ishii. SensorKnit: Architecting Textile Sensors with Machine Knitting. *3D Printing and Additive Manufacturing*, 6(1):1–11, March 2019. ISSN 2329-7662, 2329-7670. doi: 10.1089/3dp.2018.0122.

[181] p5.js. p5.js 1.0 Contributors Zine, 2020. URL http://euro-travel-example.com/index.htm.

[182] Tyler Pace, Austin Toombs, Shad Gross, Tony Pattin, Jeffrey Bardzell, and Shaowen Bardzell. A tribute to mad skill: Expert amateur visuality and world of warcraft machinima. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, page 2019–2028, New York, NY, USA, 2013. Association for Computing Machinery. ISBN 9781450318990. doi: 10.1145/2470654.2466267. URL https://doi.org/10.1145/2470654.2466267.

[183] Patrick Parzer, Florian Perteneder, Kathrin Probst, Christian Rendl, Joanne Leong, Sarah Schuetz, Anita Vogl, Reinhard Schwoediauer, Martin Kaltenbrunner, Siegfried Bauer, and Michael Haller. RESi: A Highly Flexible, Pressure-Sensitive, Imperceptible Textile Interface Based on Resistive Yarns. In *Proceedings of the 31st Annual ACM Symposium on User Interface Software and Technology*, UIST '18, pages 745–756, New York, NY, USA, October 2018. Associa-

tion for Computing Machinery. ISBN 978-1-4503-5948-1. doi: 10.1145/3242587.3242664.

[184] Nadya Peek and Neil Gershenfeld. Mods: Browser-Based Rapid Prototyping Workflow Composition. In *Proceedings of the 38th Annual Conference of the Association for Computer Aided Design in Architecture (ACADIA)*, page 6, Mexico City, Mexico, 2018. ACADIA.

[185] Nadya Peek, James Coleman, Ilan Moyer, and Neil Gershenfeld. Cardboard Machine Kit: Modules for the Rapid Prototyping of Rapid Prototyping Machines. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 3657–3668, Denver Colorado USA, May 2017. ACM. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025491.

[186] Huaishu Peng, Rundong Wu, Steve Marschner, and François Guimbretière. On-The-Fly Print: Incremental Printing While Modelling. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, pages 887–896, San Jose California USA, May 2016. ACM. ISBN 978-1-4503-3362-7. doi: 10.1145/2858036.2858106.

[187] Hannah Perner-Wilson. 2013 e-textile swatchbook exchange, 2014. https://etextile-summercamp.org/swatch-exchange/iswc-submission/.

[188] Franklin Pezutti-Dyer and Leah Buechley. Extruder-Turtle: A Library for 3D Printing Delicate, Textured, and Flexible Objects. In *Sixteenth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '22, pages 1–9, New York, NY, USA, February 2022. Association for Computing Machinery. ISBN 978-1-4503-9147-4. doi: 10.1145/3490149.3501312.

[189] M. L. V. Pitteway. Algorithm for drawing ellipses or hyperbolae with a digital plotter. *The Computer Journal*, 10(3):282–289, 01 1967. ISSN 0010-4620. doi: 10.1093/comjnl/10.3.282. URL https://doi.org/10.1093/comjnl/10.3.282.

[190] José María Pizana, Alejandro Rodríguez, Gabriel Cirio, and Miguel A. Otaduy. A Bending Model for Nodal Discretizations of Yarn-Level Cloth. *Computer Graphics Forum*, 2020. ISSN 1467-8659. doi: 10.1111/cgf.14112.

[191] Andreas Pointner, Thomas Preindl, Sara Mlakar, Roland Aigner, and Michael Haller. Knitted RESi: A Highly Flexible, Force-Sensitive Knitted Textile Based on Resistive Yarns. In *ACM SIGGRAPH 2020 Emerging Technologies*, SIGGRAPH '20, pages 1–2, New York, NY, USA, August 2020. Association for Computing Machinery. ISBN 978-1-4503-7967-0. doi: 10.1145/3388534.3407292.

[192] Andreas Pointner, Thomas Preindl, Sara Mlakar, Roland Aigner, Mira Alida Haberfellner, and Michael Haller. Knitted Force Sensors. In *Adjunct Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22 Adjunct, pages 1–3, New York, NY, USA, October 2022. Association for Computing Machinery. ISBN 978-1-4503-9321-8. doi: 10.1145/3526114.3558656.

[193] Processing. Processing Forum, 2020. URL https://discourse.processing.org/.

[194] Jie Qi and Leah Buechley. Animating Paper Using Shape Memory Alloys. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '12, pages 749–752, New York, NY, USA, 2012. Association for Computing Machinery. ISBN 978-1-4503-1015-4. doi: 10.1145/2207676.2207783. URL https://doi.org/10.1145/2207676.2207783. event-place: Austin, Texas, USA.

[195] Jie Qi, Asli Demir, and Joseph A. Paradiso. Code Collage: Tangible Programming On Paper With Circuit Stickers. In *Proceedings of the 2017 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '17, pages 1970–1977, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 978-1-4503-4656-6. doi: 10.1145/3027063.3053084. URL https://doi.org/10.1145/3027063.3053084. event-place: Denver, Colorado, USA.

[196] Andrew Quitmeyer and Hannah Perner-Wilson. Wearable Studio Practice: Design Considerations for Digital Crafting in Harsh Environments. In *Adjunct Proceedings of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2015 ACM International Symposium on Wearable Computers*, UbiComp/ISWC'15 Adjunct, pages 1285–1293,

New York, NY, USA, 2015. Association for Computing Machinery. ISBN 978-1-4503-3575-1. doi: 10.1145/2800835.2807926. URL https://doi.org/10.1145/2800835.2807926. event-place: Osaka, Japan.

[197] Raf Ramakers, Fraser Anderson, Tovi Grossman, and George Fitzmaurice. 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*, CHI '16, pages 409–419, New York, NY, USA, May 2016. Association for Computing Machinery. ISBN 978-1-4503-3362-7. doi: 10.1145/2858036.2858485. URL http://doi.org/10.1145/2858036.2858485.

[198] LLC Ravelry. Ravelry, 2023. URL https://www.ravelry.com/. Accessed 2023.

[199] Mitchel Resnick and Eric Rosenbaum. Designing for Tinkerability.

[200] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. Design Principles for Tools to Support Creative Thinking. *International Journal of Human–Computer Interaction*, 20(2):61–77, May 2006.

[201] Daniela Retelny and Pamela Hinds. Embedding Intentions in Drawings: How Architects Craft and Curate Drawings to Achieve Their Goals. In *Proceedings of the 19th ACM Conference on Computer-Supported Cooperative Work & Social Computing*, pages 1310–1322, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3592-8. doi: 10.1145/2818048.2819932. URL http://doi.acm.org/10.1145/2818048.2819932.

[202] Richard Dupont. Terminal Stage, 2008.

[203] Michael L. Rivera and Scott E. Hudson. Desktop Electrospinning: A Single Extruder 3D Printer for Producing Rigid Plastic and Electrospun Textiles. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 1–12, New York, NY, USA, May 2019. Association for Computing Machinery. ISBN 978-1-4503-5970-2. doi: 10.1145/3290605.3300434.

[204] Michael L. Rivera, S. Sandra Bae, and Scott E. Hudson. Designing a sustainable material for 3d printing with spent coffee grounds. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*, DIS '23, page

294–311, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450398930. doi: 10.1145/3563657.3595983. URL https://doi.org/10.1145/3563657.3595983.

[205] Alec Rivers, Ilan E. Moyer, and Frédo Durand. Position-correcting tools for 2D digital fabrication. *ACM Transactions on Graphics*, 31(4):88:1–88:7, July 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185584.

[206] Isabelle Rivoal and Noel B. Salazar. Contemporary Ethnographic Practice and the Value of Serendipity. SSRN Scholarly Paper ID 2321082, Social Science Research Network, Rochester, NY, 2013. URL https://papers.ssrn.com/abstract=2321082.

[207] S. Roosth. *Synthetic: How Life Got Made*. University of Chicago Press, 2017. ISBN 9780226440460.

[208] Ricarose Roque, Natalie Rusk, and Mitchel Resnick. Supporting Diverse and Creative Collaboration in the Scratch Online Community. In Ulrike Cress, Johannes Moskaliuk, and Heisawn Jeong, editors, *Mass Collaboration and Education*, Computer-Supported Collaborative Learning Series, pages 241–256. Springer International Publishing, Cham, 2016. ISBN 978-3-319-13536-6. doi: 10.1007/978-3-319-13536-6_12. URL https://doi.org/10.1007/978-3-319-13536-6_12.

[209] Daniela K. Rosner. *Critical fabulations: reworking the methods and margins of design*. Design thinking, design theory. The MIT Press, Cambridge, MA, 2018. ISBN 978-0-262-03789-1.

[210] Roxy Paine. Drawing Machine, 2001.

[211] Yannick Rémion, Jean-Michel Nourrit, and Didier Gillard. A dynamic animation engine for generic spline objects. *The Journal of Visualization and Computer Animation*, 11(1):17–26, 2000. doi: https://doi.org/10.1002/(SICI)1099-1778(200002)11:1<17::AID-VIS213>3.0.CO;2-9.

[212] Hidekazu Saegusa, Thomas Tran, and Daniela K. Rosner. Mimetic Machines: Collaborative Interventions in Digital Fabrication with Arc. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16,

pages 6008–6013, New York, NY, USA, 2016. ACM. ISBN 978-1-4503-3362-7. doi: 10.1145/2858036.2858475. URL http://doi.acm.org/10.1145/2858036.2858475.

[213] Rosa M. Sánchez-Banderas, Alejandro Rodríguez, Héctor Barreiro, and Miguel A. Otaduy. Robust eulerian-on-lagrangian rods. *ACM Trans. Graph.*, 39(4):59:59:1–59:59:10, August 2020. ISSN 0730-0301. doi: 10.1145/3386569.3392489.

[214] Ryan Schmidt and Matt Ratto. Design-to-Fabricate: Maker Hardware Requires Maker Software. *IEEE Computer Graphics and Applications*, 33(6):26–34, November 2013. ISSN 1558-1756. doi: 10.1109/MCG.2013.90. Conference Name: IEEE Computer Graphics and Applications.

[215] Adriana Schulz, Cynthia Sung, Andrew Spielberg, Wei Zhao, Robin Cheng, Eitan Grinspun, Daniela Rus, and Wojciech Matusik. Interactive robogami: An end-to-end system for design of robots with ground locomotion. *The International Journal of Robotics Research*, 36(10):1131–1147, September 2017. ISSN 0278-3649. doi: 10.1177/0278364917723465. URL https://doi.org/10.1177/0278364917723465.

[216] Susanne Seitinger, Daniel M. Taub, and Alex S. Taylor. Light Bodies: Exploring Interactions with Responsive Lights. In *Proceedings of the Fourth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '10, pages 113–120, New York, NY, USA, 2010. ACM. ISBN 978-1-60558-841-4. doi: 10.1145/1709886.1709908. URL http://doi.acm.org/10.1145/1709886.1709908.

[217] Richard Sennett. *The craftsman*. Yale Univ. Press, New Haven, 2008. ISBN 978-0-300-15119-0 978-0-300-11909-1. OCLC: 263736322.

[218] Shneiderman. Direct Manipulation: A Step Beyond Programming Languages. *Computer*, 16(8):57–69, August 1983. ISSN 1558-0814. doi: 10.1109/MC.1983.1654471.

[219] Ben Shneiderman. Creating creativity: User interfaces for supporting innovation. *ACM Transactions on Computer-Human Interaction*, 7(1):114–138, March 2000. ISSN 1073-0516. doi: 10.1145/344949.345077.

[220] Ben Shneiderman. Creativity Support Tools: A Grand Challenge for HCI Researchers. In Miguel Redondo, Crescencio Bravo, and Manuel Ortega, editors, *Engineering the User Interface*, pages 1–9. Springer London, London, 2009. ISBN 978-1-84800-135-0 978-1-84800-136-7. doi: 10.1007/978-1-84800-136-7.

[221] Sowmya Somanath, Ron Wakkary, Omid Ettehadi, Henry Lin, Armi Behzad, Jordan Eshpeter, and Doenja Oogjes. Exploring the composite intentionality of 3d printers and makers in digital fabrication. *International Journal of Design*, 16 (3):77–95, 2022.

[222] Katherine W Song, Aditi Maheshwari, Eric M Gallo, Andreea Danielescu, and Eric Paulos. Towards decomposable interactive systems: Design of a backyard-degradable wireless heating interface. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*, CHI '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450391573. doi: 10.1145/3491102.3502007. URL https://doi.org/10.1145/3491102.3502007.

[223] David J. Spencer. *Knitting Technology: A Comprehensive Handbook and Practical Guide*. CRC Press, 2001. ISBN 978-1-58716-121-6.

[224] Georg Sperl, Rahul Narain, and Chris Wojtan. Mechanics-aware deformation of yarn pattern geometry. *ACM Trans. Graph.*, 40(4):168:1–168:11, July 2021. ISSN 0730-0301. doi: 10.1145/3450626.3459816.

[225] Georg Sperl, Rosa M. Sánchez-Banderas, Manwen Li, Chris Wojtan, and Miguel A. Otaduy. Estimation of yarn-level simulation models for production fabrics. *ACM Trans. Graph.*, 41(4):65:1–65:15, July 2022. ISSN 0730-0301. doi: 10.1145/3528223.3530167.

[226] Stitchmastery. Stitchmastery – Software tools for knitters, 2023. URL https://stitchmastery.com/. Accessed 2023.

[227] Blair Subbaraman and Nadya Peek. P5.fab: Direct Control of Digital Fabrication Machines from a Creative Coding Environment. In *Designing Interactive Systems Conference*, DIS '22, pages 1148–1161, New York, NY, USA, June 2022. Association for Computing Machinery. ISBN 978-1-4503-9358-4. doi: 10.1145/3532106.3533496.

[228] Blair Subbaraman and Nadya Peek. 3d printers don't fix themselves: How maintenance is part of digital fabrication. In *Proceedings of the 2023 ACM Designing Interactive Systems Conference*, DIS '23, page 2050–2065, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9781450398930. doi: 10.1145/3563657.3595991. URL https://doi.org/10.1145/3563657.3595991.

[229] Lucy Suchman. *Human-machine reconfigurations: plans and situated actions*. Cambridge University Press, Cambridge ; New York, 2nd ed edition, 2007. ISBN 978-0-521-85891-5 978-0-521-67588-8. OCLC: ocm64592145.

[230] Lingyun Sun, Jiaji Li, Junzhe Ji, Deying Pan, Mingming Li, Kuangqi Zhu, Yitao Fan, Yue Yang, Ye Tao, and Guanyun Wang. X-bridges: Designing tunable bridges to enrich 3d printed objects' deformation and stiffness. In *Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*, UIST '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450393201. doi: 10.1145/3526113.3545710. URL https://doi.org/10.1145/3526113.3545710.

[231] Rundong Tian, Sarah Sterman, Ethan Chiou, Jeremy Warner, and Eric Paulos. MatchSticks: Woodworking through Improvisational Digital Fabrication. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, CHI '18, pages 1–12, Montreal QC, Canada, April 2018. Association for Computing Machinery. ISBN 978-1-4503-5620-6. doi: 10.1145/3173574.3173723.

[232] Rundong Tian, Vedant Saran, Mareike Kritzler, Florian Michahelles, and Eric Paulos. Turn-by-Wire: Computationally Mediated Physical Fabrication. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, pages 713–725, New Orleans, LA, USA, October 2019. Association for Computing Machinery. ISBN 978-1-4503-6816-2. doi: 10.1145/3332165.3347918.

[233] Iremnur Tokac, Herman Bruyninckx, Corneel Cannaerts, and Andrew Vande Moere. Material Sketching: Towards the Digital Fabrication of Emergent Material Effects. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI EA '19, pages 1–6, New York, NY, USA, May

2019. Association for Computing Machinery. ISBN 978-1-4503-5971-9. doi: 10.1145/3290607.3313036.

[234] Iremnur Tokac, Johan Philips, Herman Bruyninckx, and Andrew Vande Moere. Fabrication grammars: Bridging design and robotics to control emergent material expressions. *Construction Robotics*, 5(1):35–48, March 2021. ISSN 2509-811X, 2509-8780. doi: 10.1007/s41693-021-00053-0.

[235] Iremnur Tokac, Benay Gursoy, Herman Bruyninckx, and Andrew Vande Moere. Craft-inspired digital fabrication: A study of interactive robotic clay carving. In *Proceedings of the 7th Annual ACM Symposium on Computational Fabrication*, SCF '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450398725. doi: 10.1145/3559400.3562003. URL https://doi.org/10.1145/3559400.3562003.

[236] Cesar Torres and Eric Paulos. MetaMorphe: Designing Expressive 3D Models for Digital Fabrication. In *Proceedings of the 2015 ACM SIGCHI Conference on Creativity and Cognition*, C&C '15, pages 73–82, New York, NY, USA, 2015. ACM. ISBN 978-1-4503-3598-0. doi: 10.1145/2757226.2757235. URL http://doi.acm.org/10.1145/2757226.2757235.

[237] Cesar Torres, Jasper O'Leary, Molly Nicholas, and Eric Paulos. Illumination Aesthetics: Light As a Creative Material Within Computational Design. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 6111–6122, New York, NY, USA, 2017. ACM. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025466. URL http://doi.acm.org/10.1145/3025453.3025466.

[238] Jasper Tran O'Leary, Eunice Jun, and Nadya Peek. Improving programming for exploratory digital fabrication with inline machine control and styled toolpath visualizations. In *Proceedings of the 7th Annual ACM Symposium on Computational Fabrication*, SCF '22, New York, NY, USA, 2022. Association for Computing Machinery. ISBN 9781450398725. doi: 10.1145/3559400.3561998. URL https://doi.org/10.1145/3559400.3561998.

[239] Jasper Tran O'Leary, Gabrielle Benabdallah, and Nadya Peek. Imprimer: Computational Notebooks for CNC Milling. In *Proceedings of the 2023 CHI Confer-*

ence on Human Factors in Computing Systems, pages 1–15, Hamburg Germany, April 2023. ACM. ISBN 978-1-4503-9421-5. doi: 10.1145/3544548.3581334.

[240] Tiffany Tseng and Mitchel Resnick. Product Versus Process: Representing and Appropriating DIY Projects Online. In *Proceedings of the 2014 Conference on Designing Interactive Systems*, DIS '14, pages 425–428, New York, NY, USA, 2014. ACM. ISBN 978-1-4503-2902-6. doi: 10.1145/2598510.2598540. URL http://doi.acm.org/10.1145/2598510.2598540.

[241] Sherry Turkle and Seymour Papert. Epistemological Pluralism: Styles and Voices within the Computer Culture. *Signs: Journal of Women in Culture and Society*, 16(1):128–157, October 1990. ISSN 0097-9740. doi: 10.1086/494648.

[242] Joanne Turney. *The Culture of Knitting*. Berg Publishers, August 2009. ISBN 978-1-84520-592-8.

[243] Hannah Twigg-Smith and Nadya Peek. Dynamic Toolchains: Software Infrastructure for Digital Fabrication Workflows. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*, UIST '23, pages 1–20, New York, NY, USA, October 2023. Association for Computing Machinery. ISBN 9798400701320. doi: 10.1145/3586183.3606802.

[244] Hannah Twigg-Smith, Jasper Tran O'Leary, and Nadya Peek. Tools, Tricks, and Hacks: Exploring Novel Digital Fabrication Workflows on #PlotterTwitter. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pages 1–15, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8096-6. doi: 10.1145/3411764.3445653.

[245] Hannah Twigg-Smith, Yuecheng Peng, Emily Whiting, and Nadya Peek. What's in a cable? Abstracting Knitting Design Elements with Blended Raster/Vector Primitives. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*, UIST '24, pages 1–20, New York, NY, USA, October 2024. Association for Computing Machinery. ISBN 9798400706288. doi: 10.1145/3654777.3676351.

[246] Hannah Twigg-Smith, Emily Whiting, and Nadya Peek. KnitScape: Computational Design and Yarn-Level Simulation of Slip and Tuck Colorwork Knitting

Patterns. In *CHI Conference on Human Factors in Computing Systems*, CHI '24, pages 1–15, New York, NY, USA, April 2024. Association for Computing Machinery.

[247] Tom Valkeneers, Danny Leen, Daniel Ashbrook, and Raf Ramakers. StackMold: Rapid Prototyping of Functional Multi-Material Objects with Selective Levels of Surface Details. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, pages 687–699, New York, NY, USA, October 2019. Association for Computing Machinery. ISBN 978-1-4503-6816-2. doi: 10.1145/3332165.3347915. URL http://doi.org/10.1145/3332165.3347915.

[248] Richard Vallett, Denisa Qori McDonald, Genevieve Dion, Youngmoo Kim, and Ali Shokoufandeh. Toward Accurate Sensing with Knitted Fabric: Applications and Technical Considerations. *Proceedings of the ACM on Human-Computer Interaction*, 4(EICS):79:1–79:26, June 2020. doi: 10.1145/3394981.

[249] Jerry Van Aken and Mark Novak. Curve-drawing algorithms for raster displays. *ACM Trans. Graph.*, 4(2):147–169, apr 1985. ISSN 0730-0301. doi: 10.1145/282918.282943. URL https://doi.org/10.1145/282918.282943.

[250] Sonya Vasquez. Tattoo Your 3D Prints With Velocity Painting, July 2018. https://hackaday.com/2018/07/05/tattoo-your-3d-prints-with-velocity-painting/, accessed 2023.

[251] Bret Victor. Up and Down the Ladder of Abstraction, 2011.

[252] Eric von Hippel and Ralph Katz. Shifting Innovation to Users via Toolkits. *Management Science*, 48(7):821–833, July 2002. ISSN 0025-1909. doi: 10.1287/mnsc.48.7.821.2817.

[253] Ron Wakkary and Leah Maestri. The resourcefulness of everyday design. In *Proceedings of the 6th ACM SIGCHI conference on Creativity & cognition*, C&C '07, pages 163–172, New York, NY, USA, June 2007. Association for Computing Machinery. ISBN 978-1-59593-712-4. doi: 10.1145/1254960.1254984. URL https://doi.org/10.1145/1254960.1254984.

[254] Barbara G Walker. *A Treasury of Knitting Patterns*. School House Press, 1998. ISBN 978-0-942018-16-5.

[255] Mark Wheadon and Guillaume Vigneron. Velocity painting: 3d printing in style, 2022. https://www.velocitypainting.xyz/blog/, accessed 2023.

[256] Karl D.D. Willis, Cheng Xu, Kuan-Ju Wu, Golan Levin, and Mark D. Gross. Interactive fabrication: new interfaces for digital fabrication. In *Proceedings of the fifth international conference on Tangible, embedded, and embodied interaction - TEI '11*, page 69, Funchal, Portugal, 2011. ACM Press. ISBN 978-1-4503-0478-8. doi: 10.1145/1935701.1935716. URL http://portal.acm.org/citation.cfm?doid=1935701.1935716.

[257] Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. Stitch meshing. *ACM Trans. Graph.*, 37(4), jul 2018. ISSN 0730-0301. doi: 10.1145/3197517.3201360. URL https://doi.org/10.1145/3197517.3201360.

[258] Kui Wu, Hannah Swan, and Cem Yuksel. Knittable Stitch Meshes. *ACM Transactions on Graphics*, 38(1):10:1–10:13, January 2019. ISSN 0730-0301. doi: 10.1145/3292481.

[259] Kui Wu, Marco Tarini, Cem Yuksel, James McCann, and Xifeng Gao. Wearable 3d machine knitting: Automatic generation of shaped knit sheets to cover real-world objects. *IEEE Transactions on Visualization and Computer Graphics*, 28 (9):3180–3192, 2022. doi: 10.1109/TVCG.2021.3056101.

[260] Humphrey Yang, Danli Luo, Kuanren Qian, and Lining Yao. Freeform Fabrication of Fluidic Edible Materials. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*, CHI '21, pages 1–10, New York, NY, USA, May 2021. Association for Computing Machinery. ISBN 978-1-4503-8096-6. doi: 10.1145/3411764.3445097.

[261] Kentaro Yasu. Magnetic Plotter: A Macrotexture Design Method Using Magnetic Rubber Sheets. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, pages 4983–4993, Denver Colorado USA, May 2017. ACM. ISBN 978-1-4503-4655-9. doi: 10.1145/3025453.3025702.

[262] Nur Yildirim, James McCann, and John Zimmerman. Digital fabrication tools at work: Probing professionals' current needs and desired futures. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, CHI '20, page 1–13, New York, NY, USA, 2020. Association for Computing Machinery. ISBN 9781450367080. doi: 10.1145/3313831.3376621. URL https://doi.org/10.1145/3313831.3376621.

[263] Tianhong Catherine Yu and James McCann. Coupling Programs and Visualization for Machine Knitting. In *Proceedings of the 5th Annual ACM Symposium on Computational Fabrication*, SCF '20, pages 1–10, New York, NY, USA, November 2020. Association for Computing Machinery. ISBN 978-1-4503-8170-3. doi: 10.1145/3424630.3425410.

[264] Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Trans. Graph.*, 31(4), jul 2012. ISSN 0730-0301. doi: 10.1145/2185520.2185533. URL https://doi.org/10.1145/2185520.2185533.

[265] Malgorzata A Zboinska and Delia Dumitrescu. On the aesthetic significance of imprecision in computational design: Exploring expressive features of imprecision in four digital fabrication approaches. *International Journal of Architectural Computing*, 19(3):250–272, September 2021. ISSN 1478-0771. doi: 10.1177/1478077120976493.

[266] Clement Zheng, HyunJoo Oh, Laura Devendorf, and Ellen Yi-Luen Do. Sensing Kirigami. In *Proceedings of the 2019 on Designing Interactive Systems Conference*, DIS '19, pages 921–934, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 978-1-4503-5850-7. doi: 10.1145/3322276.3323689. URL https://doi.org/10.1145/3322276.3323689. event-place: San Diego, CA, USA.

[267] Clement Zheng, Bo Han, Xin Liu, Laura Devendorf, Hans Tan, and Ching Chiuan Yen. Crafting interactive circuits on glazed ceramic ware. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*, CHI '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN

9781450394215. doi: 10.1145/3544548.3580836. URL https://doi.org/10.1145/3544548.3580836.

[268] Junqiu Zhu, Adrian Jarabo, Carlos Aliaga, Ling-Qi Yan, and Matt Jen-Yuan Chiang. A realistic surface-based cloth rendering model. In *ACM SIGGRAPH 2023 Conference Proceedings*, SIGGRAPH '23, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400701597. doi: 10.1145/3588432.3591554. URL https://doi.org/10.1145/3588432.3591554.

[269] Amit Zoran and Joseph A. Paradiso. FreeD: A Freehand Digital Sculpting Tool. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '13, pages 2613–2616, New York, NY, USA, 2013. ACM. ISBN 978-1-4503-1899-0. doi: 10.1145/2470654.2481361. URL http://doi.acm.org/10.1145/2470654.2481361.

Appendix A

Example Toolchain Module Implementations

Axidraw Plotting Module Implementation

To illustrate a module with a more complex full-stack implementation, we include the full implementation of the AxidrawPlot module. These files are stored in the same folder and detected automatically by our module library. Other than installing the pyaxidraw library, there are no additional development tasks for this module to be used in the system.

```
{
  "displayName": "AxiDraw Plot",
  "inports": {
    "svg": {
      "displayName": "svg"
    }
  },
  "outports": {
    "preview": {
      "displayName": "preview"
    }
  },
  "state": {}
}
```

Listing 6 *The JSON configuration for the AxiDraw Plot module specifies that it has one inport for accepting SVG and one outport for outputting a toolpath preview.*

```python
from planager.ToolModule import ToolModule
from pyaxidraw import axidraw

class AxidrawPlot(ToolModule):
    # Runs when the module is added to the environment
    def setup(self):
        self.ad = axidraw.AxiDraw()

    # Specifies common AxiDraw settings
    def plot_common(self):
        self.ad.options.model = 2
        self.ad.options.reordering = 2

    # Generates a preview of the input SVG's toolpath
    def preview(self, arg):
        if not self.ad or not self.inports["svg"]:
            return

        self.ad.plot_setup(self.inports["svg"])
        self.ad.options.preview = True
        self.plot_common()

        # Sends the toolpath SVG to the preview outport
        output_svg = self.ad.plot_run(True)
        self.outports["preview"] = output_svg

    # Plots the SVG currently in the SVG inport
    def run(self, arg):
        if not self.ad or not self.inports["svg"]:
            return

        self.ad.plot_setup(self.inports["svg"])
        self.plot_common()
        self.ad.plot_run()
```

Listing 7 *The AxidrawPlot module wraps the* ***plot_run*** *method provided by the AxiDraw Python library. With only a small amount of code, the AxiDraw can accept dataflow input and be run directly from the toolchain configuration interface.*

```
import { html, css } from "lit";
import { Tool } from "planager/ToolModule";

export default class AxidrawPlot extends ToolModule {
  static styles = css`
    .button {
      background-color: var(--blue);
      color: var(--text-light);
      padding: 0.2rem 0.5rem;
      text-align: center;
    }
    .button:hover {
      background-color: var(--workspace-background);
      cursor: pointer;
    }
  `;
  render() {
    return this.renderModule(html`
      <div
        class="button"
        @click=${(e) => this.api.runMethod("preview")}>
        Generate preview
      </div>
      <div
        class="button"
        @click=${(e) => this.api.runMethod("run")}>
        Run!
      </div>
    `);
  }
}
```

Listing 8 *The AxidrawPlot web component renders two buttons that trigger associated Python methods when pressed. The interface is defined with HTML and CSS template literals. Our system automatically handles websocket communication between the web component and Python implementation.*

Mapbox Module Implementation

```
{
  "displayName": "Map",
  "inports": {
    "query": {
      "displayName": "query"
    }
  },
  "outports": {
    "geojson": {
      "displayName": "geojson"
    },
    "bounds": {
      "displayName": "bounds"
    },
    "view": {
      "displayName": "view"
    }
  },
  "state": {
    "lat": 47.658,
    "long": -122.2925,
    "zoom": 10
  }
}
```

Listing 9 *The JSON configuration for the Mapbox module defines one inport that accepts a JSON query, which is used to select features to display. It defines outports for GeoJSON, current lat/long bounds, and the viewport ratio. It manages three state variables, latitude, longitude, and zoom, which can be used to save and reload the map at a particular area.*

```
from planager.ToolModule import ToolModule

class GeoMap(ToolModule):
    def set_geojson(self, geojson):
        self.outports["geojson"] = geojson

    def set_bounds(self, bounds):
        self.outports["bounds"] = bounds

    def set_view(self, view):
        self.outports["view"] = view
```

Listing 10 *The Python portion of the Mapbox module sends data to outports. This listing shows how lightweight this implementation can be.*

```
import { html, css } from "lit";
import { ToolModule } from "planager/ToolModule";
import { createRef, ref } from "lit/directives/ref.js";

import mapboxGl from "mapbox-gl";

export default class GeoMap extends ToolModule {
  static styles = css`
    #map-container {
      height: 30rem;
      width: 30rem;
      resize: both;
    }
  `;
  mapContainer = createRef();
  map;

  firstUpdated() {
    if (this.map) return; // Initialize map only once

    this.map = new mapboxGl.Map({
      container: this.mapContainer.value,
      style: defaultStyle,
      center: [this.state.long, this.state.lat],
      zoom: this.state.zoom,
      projection: "mercator",
    });

    // Callback to update module state when map is moved or zoomed
    this.map.on("move", () => {
      this.state.long = this.map.getCenter().lng.toFixed(4);
      this.state.lat = this.map.getCenter().lat.toFixed(4);
      this.state.zoom = this.map.getZoom().toFixed(2);
```

```
    });

    // Callback to request GeoJSON features when map idles and send data to outports
    this.map.on("idle", (e) => {
      const features = this.map.queryRenderedFeatures();

      this.api.runMethod("set_geojson", features);

      const bounds = this.map.getBounds();
      this.api.runMethod("set_bounds", bounds);

      const map = this.mapContainer.value;
      this.api.runMethod("set_view", {
        width: map.clientWidth,
        height: map.clientHeight,
      });
    });

    // If the map container changes size, call the mapbox map resize method
    const observer = new ResizeObserver(() => {
      this.map.resize();
    });
    observer.observe(this.mapContainer.value);
  }

  // The render method runs when inports are updated
  render() {
    if (this.inports.config) {
      try {
        this.map.setStyle(this.inports.config);
      } catch {
        console.log("Invalid map style");
      }
    }
    return this.renderModule(html`
        <link
          href="https://api.mapbox.com/mapbox-gl-js/v2.10.0/mapbox-gl.css"
          rel="stylesheet" />
        <div>
          <div
            id="map-container"
            ${ref(this.mapContainer)}></div>
        </div>
    `);
  }
};
```

Listing 11 *The MapboxGL module uses the Mapbox API to display an interactive map. It periodically queries the API for a set of GeoJSON features and sends them to the outport.*

Appendix B

Knitting Overview

This section provides a high-level overview of the domain of knitting and definitions for terms used in this dissertation. For a comprehensive text on knitting technology, we refer to Spencer [223].

Knitting refers to the process of manipulating yarn into fabric via interlooping, in which loops of yarn are pulled through other loops. Knitting is often compared (and confused) with *weaving*, where fabric is formed by interlacing two sets of yarns at right angles. Both techniques produce different attributes in the fabric with different applications: knits are typically more breathable and stretchier (T-shirts, socks), while wovens are firm and stable (denim, bedsheets, towels).

Hand knitting is at least 1500 years old and continues to be extremely popular today. Hand knitting is typically performed on straight pointed needles. As a general rule, hand knitting has a far larger possibility space than machine knitting, so anything that can be knit on a machine can be knit by hand. However there are practical tradeoffs in terms of speed, pattern complexity, and consistency.

Knitting machines also use needles, but each needle holds only a small number of loops at a time (typically only one or two). Knitting machine needles are shaped like a small hook with a latch that can open and close. On the machine, needles are arranged into one or more *beds*. In this paper we are concerned with *V-bed knitting machines*, in which two beds are oriented in an upside-down V shape, with the needle hooks facing each other at the top. Other classes of knitting machines have different bed configurations. For example, circular knitting machines have a circular bed.

A *carriage* moves along the needle bed to guide the yarn and actuate the needles according to the pattern. Typically, machines have multiple *yarn feeders* which the

carriage will pick up and drop off according to which yarn is in use. During the carriage movement, the type of stitch performed by each needle is determined by how far it moves out and in. To knit a stitch, when the needle moves out the previously-held loop slides back behind the latch, and a new loop of yarn is laid into the open hook. Then, as the needle moves back in, the previous loop slides forward, pushes the latch closed, and drops over the closed latch.

Glossary of terms

stitch
: Used interchangeably to refer either to: a single loop of yarn embedded in fabric, the unit operation which produced it, or to a repeating texture pattern (e.g. “seed stitch”).

knit
: Refers to fundamental knitting operation in which a loop of yarn is pulled through an existing loop. The yarn can be pulled through a loop from either side, producing either a knit (which resembles a “v”) or a purl (resembling a small bump). In machine knitting, these operations are often referred to by the bed used to create them (“front knit” or “back knit”), while hand knitters refer to them as “knits” and “purls”.

tuck
: An operation where a loop of yarn is added to a needle without pulling it through an existing loop (if present)

slip, skip, miss
: An operation where yarn is passed behind a loop without being pulled through it. In machine knitting, the latch hook needle does not extend to pick up a new loop. In hand knitting, the loop is passed without modification from one straight needle to another. In both cases, the extra yarn produces a *float*.

transfer
: An operation where loops are moved from one needle to another, often used to create textures like lace or cables or to shape the fabric.

float

A loose bit of yarn not directly worked into the fabric structure. Typically there is an upper limit to the length of a float, as floats which are too long may snag on fingers or jewellery.

chart

A visual representation of a knitting pattern, in which symbols and colors are arranged on a grid to communicate the placement of stitches, yarns, and techniques across the rows and columns of the fabric.

colorwork

An umbrella term describing knitting techniques where different colors of yarn are incorporated into a single piece to create color patterns.

stranded or fair-isle colorwork

A colorwork technique for working two or more colors of yarn in the same row by floating the non-showing yarn across the back of the fabric.

intarsia

A colorwork technique where each separate block of color in a row is knit by a dedicated yarn. This means that yarns are not floated across the back of the work (as they are in stranded colorwork), making it an ideal technique for larger, solid-colored areas.

cable

A texture effect where loops are “crossed”, which can create the appearance of twisted rope on the fabric surface. Knitting machines use transfers to rearrange the loops, and hand knitters typically use an extra needle (a “cable needle”). Cable knitting is a family of techniques for producing different effects via crossed stitches, including braids, twists, and lattices.

rib A texture created by alternating knit and purl stitches in vertical columns. This produces a very stretchy fabric commonly used for cuffs, collars, and hems.

lace A technique that incorporates patterns of holes or gaps, often created by using transfers to move loops to create holes and form intricate designs.

Appendix C

Pseudocode for Boundary and Path Rasterization

Function `PlotPath`($p0$, $p1$, $block$, $offset$, $mode$)**:**

$[x0, y0] \leftarrow p0$;
$[x1, y1] \leftarrow p1$;
$dx \leftarrow abs(x1 - x0)$;
$sx \leftarrow x0 < x1 ? 1 : -1$;
$dy \leftarrow -abs(y1 - y0)$;
$sy \leftarrow y0 < y1 ? 1 : -1$;
$error \leftarrow dx + dy$;
$last \leftarrow [x0, y0]$;
while *True* **:**
  switch $mode$ **do**
    case *"overlap"* **do**
      `DrawBlock`($block, x0 + offset.x, y0 + offset.y$)
    case *"tiled"* **do**
      if $abs(x0 - last.x) \geq block.width$ *or* $abs(y0 - last.y) \geq block.height$ **:**
        `DrawBlock`($block, x0 + offset.x, y0 + offset.y$);
        $last \leftarrow [x0, y0]$;
    case *"dx"* **do**
      if $x0 \neq last.x$ **:**
        `DrawBlock`($block, x0 + offset.x, y0 + offset.y$);
        $last \leftarrow [x0, y0]$;
    case *"dy"* **do**
      if $y0 \neq last.y$ **:**
        `DrawBlock`($block, x0 + offset.x, y0 + offset.y$);
        $last \leftarrow [x0, y0]$;
  if $x0 = x1$ *and* $y0 = y1$ **:** break;
  $e2 \leftarrow 2 \times error$;
  if $e2 \geq dy$ **:**
    if $x0 = x1$ **:** break;
    $error \leftarrow error + dy$;
    $x0 \leftarrow x0 + sx$;
  if $e2 \leq dx$ **:**
    if $y0 = y1$ **:** break;
    $error \leftarrow error + dx$;
    $y0 \leftarrow y0 + sy$;

Function `RasterizePath`(*path*)**:**

$points, stitchBlock, yarnBlock,$ *offset*$, mode \leftarrow path$;
for *each segment pair* $p0, p1$ *in* $points$ **:**
  `PlotPath`($p0$, $p1$, $stitchBlock$, *offset*, $mode$);
  `PlotPath`($p0$, $p1$, $yarnBlock$, *offset*, $mode$);

Algorithm 1: Path rasterization

```
Function AddEdge(EdgeTable, [x1, y1] ← p1, [x2, y2] ← p2):
    if y1 = y2: return;
    if y1 > y2:
        [x1, x2] ← [x2, x1];
        [y1, y2] ← [y2, y1];
    dx ← (x2 − x1)/(y2 − y1);
    edge ← edge {
        x ← x1 + (dx/2);
        dx ← dx;
        yMin ← y1;
        yMax ← y2;
        xLast ← x1;
    };
    EdgeTable.insert(edge);
Function RasterizeBoundary(boundary):
    points, stitchBlock, yarnBlock, offset, shaping ← boundary;
    EdgeTable ← [];
    for i ← 0 to points.length:
        AddEdge(EdgeTable, points[i], points[(i + 1)%points.length])
    EdgeTable ← EdgeTable sorted on x;
    ActiveEdgeTable ← [];
    y ← 0;
    while there are edges left to process:
        move edges from EdgeTable to ActiveEdgeTable where yMin = y;
        for i ← 0; i < ActiveEdgeTable.length; i ← i + 2:
            left ← ActiveEdgeTable[i];
            right ← ActiveEdgeTable[i + 1];
            x0 ← round(left.x);
            x1 ← round(right.x);
            if x0 = x1: continue;
            StitchBlockY ← (y - offset.y) % stitchBlock.height;
            YarnBlockY ← (y - offset.y) % yarnBlock.height;
            for x ← x0 to x1:
                StitchBlockX ← (x - offset.x) % stitchBlock.width;
                YarnBlockX ← (x - offset.x) % yarnBlock.width;
                stitch ← stitchBlock.at(StitchBlockX, StitchBlockY);
                yarn ← yarnBlock.at(YarnBlockX, YarnBlockY);
                if stitch ≠ TRANSPARENT: DrawStitch(stitch, x, y);
                if yarn ≠ TRANSPARENT: DrawYarn(yarn, x, y);
                if shaping > 0:
                    ShapeLeft(left.xLast − x1);
                    ShapeRight(right.xLast − x2);
                left.xLast ← x1;
                right.xLast ← x2;
        remove edges from ActiveEdgeTable where yMax = y;
        foreach edge in ActiveEdgeTable do
            edge.x ← edge.x + edge.dx;
        y ← y + 1;
```

Algorithm 2: Boundary Rasterization