Finding the Grammar of Generative Craft

by

Shiqing He

A dissertation submitted in fulfillment
of the requirements for the degree of
Doctor of Philosophy
(Information)
in The University of Michigan
2021

Doctoral Committee:

Professor Eytan Adar, Chair
Professor Sophia Brueckner
Professor Ron Eglash
Professor Matthew Kay

Shiqing He

heslicia@umich.edu

ORCID ID: 0000-0002-6547-8481

For my sister Shiyuan He 何诗源

ACKNOWLEDGEMENTS

I started my Ph.D. journey in the fall of 2015, intending to search for a way to combine my passion for art and technology. However, I quickly realized that the journey was far more challenging than my expectation. Without the support of many, it would have been impossible to navigate through all the confusion, detours, and roadblocks.

I want to thank my advisor, Eytan Adar, for being the best advisor I could possibly imagine. I appreciate that Eytan would see potentials in my random (and sometimes bizarre) project ideas. In the past six years, I have tried many different research directions. The tremendous intellectual freedom that Eytan offered me eventually led me to my current research direction. The opportunity to pursue my ideal research direction is truly a privilege that I am grateful for. I also want to thank my dissertation committee members, prof. Sophia Brueckner, prof. Ron Eglash, and prof. Matthew Kay, for their support of this work. I want to thank prof. Paul Resnick, prof. Paramveer Dhillon, prof. Ceren Budak, prof. Erin Krupka, prof. David Jurgens, and prof. Cliff Lampe for offering me help at different stages of my Ph.D. journey. In addition, I am honored to have received dissertation funding support from the ArtsEngine, Rackham Graduate School, and the School of Information

(UMSI).

I am incredibly fortunate to become friends with a fantastic group of colleagues from the School of Information. I want to thank Cindy Lin, Penny (Diep) Trieu, Hariharan Subramonyam, Joey Chiao-Yin Hsiao, and the rest of my Ph.D. cohort for all the fun memories we created together. I would also like to thank the following friends & colleagues for their support in the past six years: Teng Ye, Youyang Hou, Elsie Lee, John Joon Young Chung, Wei Ai, Shiyan Yan, Chuan-Che Jeff Huang, Yingzhi Liang, and Xin Rong.

Lastly, I am grateful for all the love and support my friends and family have given me. Particularly, I want to thank Yifan Wang, Hao Wang, Juejin Lu, Juexuan Lu, and the rest of my fantastic gaming friends for all the laughter we shared. I thank Qiao Mu, Vivi Zhang, Linjun Zhang, Xiaolin Wang, Hristina Milojevic, Ziyong Lin, and Zhaoqing Shen for all the joy they have given me. I am deeply thankful for my mother, Limei Du, and my father, Pei He. Thanks for always supporting my decisions and believing in me. I am grateful to have Doudou and Erkuai in our lives. Finally, I can never thank my sister Shiyuan He enough. She is my true inspiration for life.

TABLE OF CONTENTS

LIST OF FIGURES

Figure

ABSTRACT

Art and craft design is challenging even with the assistance of computer-aided design tools. Despite the increasing availability and intelligence of software and hardware, artists continue to find gaps between their practices and tools when designing physical craft artifacts. In many craft domains, artists need to acquire domain knowledge and develop skills in design-aid tools separately. Despite their power and versatility, generic design tools pose various challenges, such as requiring workarounds for specific crafts and having steep learning curves. Compared to generic design-aid tools, craft-specific systems can offer reasonable solutions to specific design tasks because they can offer domain-specific support. Nevertheless, craft-specific tools often have limited flexibility.

In this dissertation, I introduce Grammar-driven Craft Design Tools (GCDTs), which explicitly embed and utilize craft domain knowledge (i.e., "grammar" of the craft) as their primary mechanisms and interfaces. Like other types of information, craft knowledge is processable and organizable data. In this dissertation, I develop and examine a framework to document, process, preserve, and utilize craft domain knowledge. GCDTs are craft-specific tools. By explicitly embedding and utilizing

craft domain knowledge, GCDTs bridge the gap between design-aid tools and craft domain knowledge. GCDTs also have additional benefits such as supporting generative design, facilitating learning, and preserving domain knowledge. This dissertation gives an overview of how the next generation of design-aid tools can help artists find their creative expressions. It presents the GCDT framework and introduces three GCDTs developed for distinct domains. *InfiniteLayer* assists the design of multilayer sculpture, a form of sculpture made with layers of material. Then, *MarkMakerSquare* helps designers to invent unconventional and creative mark-making tools using various fabrication strategies. Lastly, *ThreadPlotter* supports the design and fabrication of plotter-based delicate punch needle embroidery.

CHAPTER I

Introduction

1.1 Overview

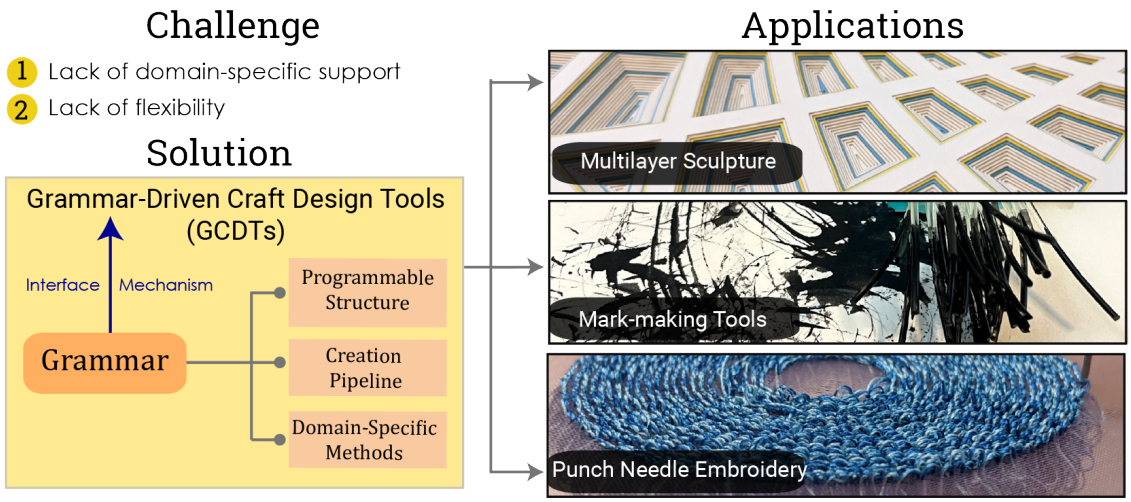

Figure 1.1: Dissertation Overview

Physical Art & Craft design is challenging even with the assistance of computer-aided design tools. Despite the increasing availability and intelligence of software and hardware, artists continue to find gaps between their practices and tools. Identifying

artists' needs and providing creative solutions are critical research challenges across disciplines such as human-computer interaction, computer graphics, design, and engineering. Intending to support the design and fabrication of physical creations by augmenting creative practices with software and hardware innovation, I examine a fundamental question in the domain of creativity support tools (CSTs):"*How to make tools for artists?*" Specifically, I focus on physical art & craft domains that are primarily designed and made by hand.

In many physical art & craft domains, designers often need to acquire domain knowledge and develop skills in design-aid tools separately. For instance, a crochet pattern designer not only need to learn about crochet, they also need to acquire expertise in pattern drafting tools. Despite their power and versatility, generic design tools pose various challenges, such as requiring workarounds for specific crafts and having steep learning curves. By leveraging theories and technological advances in human-computer interaction, information science, and digital fabrication, this dissertation examines a new direction for making craft-design tools: instead of building generic design systems, I design, develop, and test craft-specific tools that embed craft's domain knowledge (i.e., "grammar" of the craft). Grammar-driven Craft Design Tools (GCDTs) bridge the gap between design-aid tools and craft domain knowledge. They also have additional benefits such as supporting generative design, facilitating learning, and preserving domain knowledge.

Like other types of information, craft knowledge is processable and organizable data. In many craft domains, this knowledge scatters among books, tutorial videos, and online discussions. In this dissertation, I propose and examine a framework to

document, process, preserve, and utilize craft domain knowledge. I develop strategies for organizing craft knowledge into human-/machine-digestible, extensible, and programmable structures through my dissertation work. With a consistent framework for managing craft-specific domain knowledge, design tools can support artisans by embedding craft knowledge explicitly.

In this dissertation, **I present the GCDT framework by focusing on three major characteristics that define GCDTs: grammar-driven, extendable, and supporting fabrication-ready designs**. For each of these characteristics, I demonstrate a GCDT developed for a specific craft domain. Through these three distinct craft domains, this dissertation investigates the potential for using the GCDT framework to support art-making in a diverse range of Art & Craft domains (Figure 1.1).

1.2 Making Tools For Craft Designers

In this section, I provide background information and motivation for this research. Specifically, I start by defining the target scope of this research and providing a narrowed characterization for Art & Craft (Section 1.2.1). Then, I present an overview of existing craft design-aid tools (Section 1.2.2). Lastly, I discuss challenges associated with current craft design-aid systems in Section 1.2.3.

1.2.1 Creative Processes that Produce Physical Artifacts

While the definition of "craft" has shifted over time and differs under contexts (*Shiner* (2012)), craft-making is tightly associated with the manipulation of mate-

rial. Among many attempts to define "craft", art historians, anthropologists, and human-computer interaction (HCI) researchers offer distinctive perspectives. Historically, "craft" was conceived as a set of disciplines (e.g., pottery, glass painting, and metalwork) or a generic process and practice such as teaching, cooking, and parenting (*Shiner* (2012)). Identifying the boundary between "art" and "craft" is a challenging task. Instead of defining such a boundary, *Shiner* (2012) suggests that:" the boundary between art and craft conceived as a set of disciplines defined by material and techniques has not become blurred, it has all but disappeared." Studies from anthropology (e.g., *Adamson* (2010)) also suggest that "craft" should have a fluid definition instead of considered as an art form or a fixed set of disciplines. *Adamson* (2010) examines culture contexts where the term "craft" is used and suggests that "craft should be seen in fluid and relative, rather than limiting and categorical, terms."

In comparison to these perspectives, craft and craft-making in HCI research are more connected with the practice of producing designed artifacts. Because the rise of craft-focused research in HCI is connected with the maker-culture and fabrication technologies (*Nitsche et al.* (2014); *Nitsche and Weisling* (2019)), many craft-related research in HCI focus on providing design solutions for specific disciplines (see Section 1.2.2), and understanding how new technology impact traditions (e.g., *Rosner and Ryokai* (2009)). Notably, *Frankjær and Dalsgaard* (2018) discussed four types of craft in HCI research, and here I summarize:

- Hybrid craft: creations that combine physical and digital material.
- Digital Craft: design and fabrication of physical artifact using digital tools.

- Computational Craft: using computer-generated patterns for realizing handcraft.
- Technocraft: craft with technological objects.

Because of the vast research space associated with Art & Craft, I limit the scope of this dissertation by providing a narrowed characterization for Art & Craft. In this dissertation, **I focus on examining Art & Craft making as creative processes that produce physical artifacts.** Using these criteria, I can target Art & Craft domains without drawing clear boundaries using functionality, aesthetic value, or technologies involved. For example, an embroidery piece could be a functional artifact and a decorative object at the same time. Embedding electronics in this embroidery piece would make this piece an example of a hybrid craft. Therefore, this embroidery piece is an example that sits in the scope of this dissertation. In addition, this criteria directs attention to domains strongly associated with manipulating and transforming physical material. In this case, I exclude digital-focused disciplines such as music, film-making, and digital painting from this research. In the remainder of this dissertation, "Art & Craft" and "craft" both refer to the narrowed scope unless otherwise noted.

1.2.2 Design-Aid Tools for Art & Craft

There exists a wide range of Art & Craft design tools and various ways to categorize them. In creativity support tools (CST) research, several studies provide frameworks for understanding CSTs. Existing studies that analyze CST focus on

computer-enabled systems that support a wide range of creative processes. For example, *Frich et al.* (2018, 2019) track the evolution of CSTs in the HCI research community and synthesize patterns that emerged in these tools. *Remy et al.* (2020) analyzes how CSTs are evaluated. *Chung et al.* (2021) surveys over a hundred CST-related research papers to contribute a taxonomy for describing and discussing CSTs. Specifically, *Chung et al.* (2021) provides a framework to "understand the intersection of technologies, interactions, roles, and users in shape art-making CSTs." For example, CSTs take different roles when supporting their designers. Some tools support users with artistic visions and ideas (the *Vision* role). Some tools assist users by providing expertise and reducing labor (the *Skill* role) (*Chung et al.* (2021)).

1.2.2.1 Categorizing Craft Design-Aid Tools Through Three Dimensions

CST-focused survey research like *Chung et al.* (2021); *Remy et al.* (2020); *Frich et al.* (2018) and *Frich et al.* (2019) target a much more extensive range of tools than the Art & Craft scope I target in this dissertation. For example "art-making" in *Chung et al.* (2021) is a superset of the Art & Craft domains that I defined in Section 1.2.1 [1]. To provide an overview of existing craft design-aid tools that fits the scope of this dissertation, I utilize three binary dimensions to capture different characteristics of craft design-aid tools (see Figure 1.2).

Physical vs. Digital

In craft domains traditionally made by hand, the design tool associated with these crafts is also physical objects. For instance, rulers, compasses, and stencils are

[1] They use the term "art-making" to indicate activities for making creative aesthetic artifacts. The artifacts do not need to have physical forms.

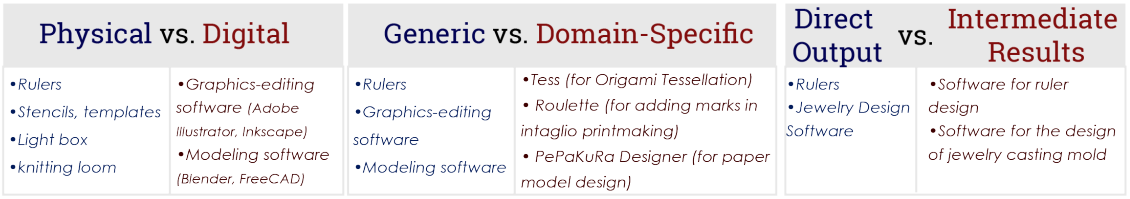

Figure 1.2: To provide an overview of existing craft design-aid tools that fits the scope of this dissertation, I utilize three dimensions to capture different characteristics of craft design-aid tools

among the most prevalent physical design-aid tools. In addition to these "universal" design-assist tools, many disciplines also have specialized tools to support their design processes. For example, ceramic artists can use carving tools to shape their creations. Printmakers use a combination of magnesium carbonate powder and black paper to visualize mezzotint designs[2]. With the development of technology, computer-based tools have become an integral part of many craft disciplines. For instance, graphic editing software such as *Adobe Illustrator*[3] and *INKSCAPE*[4] have been widely used in many craft domains, ranging from stencil design to printmaking. 3D modeling software such as *Blender*[5] and *SOLIDWORKS*[6] also enable a wide range of craft creation.

Generic vs. Domain-Specific Tools

We can also categorize craft-aid tools base on their target disciplines. While drafting and graphics editing tools support an extensive range of craft design tasks, some tools are designed to help a specific craft or even a minor step in the creation pro-

[2]Mezzotint is an intaglio printmaking technique.
[3]`https://www.adobe.com/products/illustrator.html`
[4]`https://inkscape.org/`
[5]`https://www.blender.org/`
[6]`https://www.solidworks.com/`

cess. For instance, *PePaKuRa Designer*[7] is a software made for paper model design. Given a 3D model, *PePaKuRa Designer* automatically unfolds the 3D polygon-mesh model and appends additional utility structures for paper model designs. Besides being a domain-specific software that targets paper-model building, *PePaKuRa Designer* also focuses on a specific design step within the creation pipeline. Instead of building full-fledged modeling software, *PePaKuRa Designer* takes a more focused strategy and only supports the unfolding of 3D models. Within the fields of HCI and computer graphics, many research projects produce domain-specific systems. To name a few, *Igarashi et al.* (2016b) contributes a tool to design Iris folding patterns[8]; *Oh et al.* (2015) provides a system to design paper mechanics; *Torres et al.* (2016) offers a toolkit to make molds for jewelry design. Section 1.2.2 provides a detailed review of these systems.

Direct Output vs. Intermediate Results

The last dimension captures the approach that design-aid tools take to support their users. Helping designers generate the final output is one strategy that many tools/systems adopt. For example, tools like *Igarashi et al.* (2016a) assist their users in making jewelry designs directly. The outputs of these tools are the target artifacts in these cases. In contrast, tools like *Iarussi et al.* (2015a); *Torres et al.* (2016) and *Igarashi* (2011) take a different strategy. Instead of supporting the design of the final artifact, these tools support the design of intermediate results. The intermediate results can take various forms. For example, *Iarussi et al.* (2015a) and *Torres et al.* (2016) generate molds for jewelry design instead of jewelry pieces. *Igarashi* (2011)

[7] `https://tamasoft.co.jp/pepakura-en/`

[8] Iris folding is a paper craft that makes patterns with paper strips.

and *Igarashi et al.* (2012) generate instruction for rhino stone patterns and bead sculptures so that artisans can manually construct their artifacts.

These three dimensions summarize high-level design decisions that craft system developers need to make while building their tools. For example, generic tools and domain-specific tools assist their users in drastically different ways. From a tool designer's perspective, it is critical to examine how these design decisions impact their systems, which leads to an important question: "how to support craft design?"

1.2.2.2 Craft Research in HCI

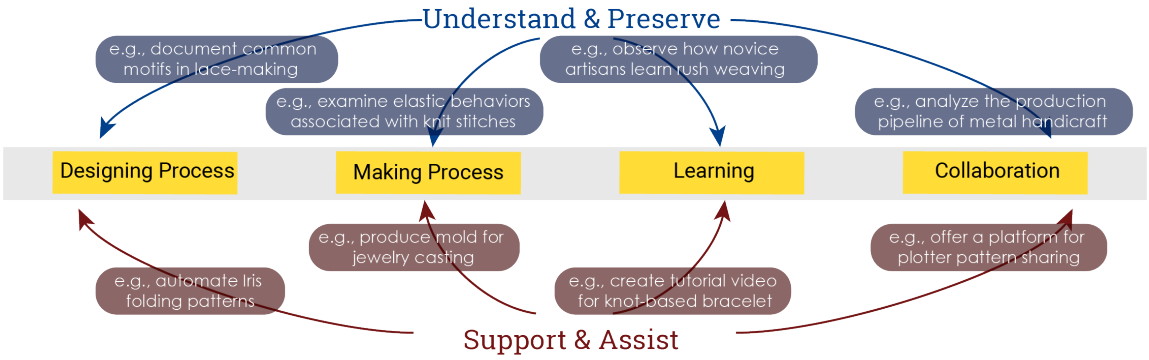

Figure 1.3: Two major approaches that existing studies take to support craft design in the field of HCI.

To get a closer look at approaches that current research and systems take to support Art & Craft design, I surveyed recent craft-related studies within the field of HCI. Overall, existing work in the field support craft design using two major approaches: 1) Understand & Preserve, and 2) Support & Assist. In addition to these high-level approaches, there are four major aspects that existing studies focus on: 1) Designing Process, 2) Making Process, 3) Learning, and 4) Collaboration.

Understand & Preserve

One way to support craft design is by investigating the craft domain and documenting domain knowledge. Obtaining an understanding of a craft is an essential requirement for supporting craft design. Formulating this understanding into a set of rules is a common contribution of craft-related research. For example, *Noel* (2015) provides a demonstrative example of craft knowledge preservation in the domain of Trinidad Carnival wire binding sculpture. By conducting interviews, site visits, and observations, the author presents rules that synthesize the craft.

Knowledge required to make the craft is undoubtedly an essential type of information to preserve. Many existing projects focus on revealing and summarizing making processes of different craft domains. For instance, *Irvine and Ruskey* (2014) identifies a mathematical model for bobbin lace, which is the traditional way to make lace. *Markande and Matsumoto* (2020) documents topological structures behind stitches in knitting. Research like *Zhang et al.* (2015) and *Kryven and Fourquet* (2013) provide additional examples in the domain of bead-/block-based sculpture design. In addition to the knowledge related to the production of artifacts, some studies investigate how artists make a new design. Studies in this category enhance understanding of the craft by preserving standard design practices and documenting design strategies. For example, *Noel* (2015) presents a grammar for wire-binding sculpture in the Trinidad Carnival. *Kono and Watanabe* (2017) documents various sewing techniques that can alter textile shapes. *Cromwell* (2008) examines Celtic interlaced ornaments, which are artifacts constructed using small knots.

In addition to revealing knowledge related to making and designing a craft,

projects like *Tung* (2012) and *Mahato et al.* (2019) also examine how novice practitioners learn or collaborate in this craft domain. Specifically, *Tung* (2012) documents the material used for teaching rush weaving. *Mahato et al.* (2019) focuses on how artisans and craft designers work together to develop new metal handicraft designs.

Support & Assist

Besides enhancing our understanding of a craft domain, research projects also produce tools that can aid in designing, making, and learning the craft. Studies that go beyond understanding a craft domain often come with digital tools that assist the design. More specifically, these tools normally address design challenges from the following aspects:

- Support the designing of artifacts. E.g., *Iarussi et al.* (2015a) assists the design of wire-wrapped jewelry.
    - Support the designing of artifacts through programming and robots. E.g., *Scalera et al.* (2019) presents a robot for watercolor painting.
- Support the making of artifacts. E.g., *Igarashi* (2011) helps designers to create stencils for rhinestone decorations, which is a pattern-making art form that features round rhinestones.
- Support the learning of a craft. E.g., *Oh et al.* (2015) documents techniques in the design of linkage-based paper toys and helps designers to learn the design space of this craft.

Figure 1.3 summarizes these two approaches and four aspects that the current research target. It also provides examples to illustrate how different projects/systems

support craft design. These approaches and aspects are not exclusive to each other, as a tool can cover multiple aspects. Table 1.1 lists a selection of existing studies and their approaches. I adopt both approaches in this dissertation. GCDTs provide mechanisms to preserve and understand craft information through grammar. At the same time, GCDTs support the design and making of physical artifacts.

Paper	Domain	Understanding				Supporting			
		M.	D.	C.	L.	M.	D.	C.	L
Irvine and Ruskey (2014)	Bobbin Lace	•							
Markande and Matsumoto (2020)	Knotty Knits	•							
Noel (2015)	Trinidad Carnival Wire Binding	•	•						
Kono and Watanabe (2017)	3D shape construction using fabric	•	•						
Cromwell (2008)	Celtic Interlaced Ornament	•	•						
Igarashi et al. (2012)	Bead Sculpture	•				•			
Zhang et al. (2015)	Mini Block Sculpture	•				•			
Kryven and Fourquet (2013)	Knitting	•				•			
Tung (2012)	Rush Weaving	•	•		•				
Mahato et al. (2019)	Metal Handicraft	•	•	•					
Igarashi et al. (2016b)	Iris Folding Pattern	•	•			•			
Paczkowski et al. (2018)	Paper-based 3D modeling	•	•			•			
Igarashi (2019)	Band Weaving	•	•			•			
Igarashi et al. (2016a)	Necklace Design	•	•			•			
Skouras et al. (2015)	Interlocking Objects	•				•	•		
Igarashi et al. (2009)	Cover Design	•	•			•			
Wu et al. (2018)	Knitting	•				•	•		
Zheng and Nitsche (2017)	Ceramics		•	•			•	•	
Scalera et al. (2019)	Robot-based watercolor painting	•	•				•		•
Iarussi et al. (2015a)	Wire Wrapped Jewelry	•	•			•	•		
Igarashi (2011)	Rhinestone Sculpture	•	•			•	•		
Torres et al. (2016)	Jewelry Making	•	•			•	•		
Oh et al. (2015)	Paper Mechanics	•	•			•	•		•

Table 1.1: Selected Craft-related Research Projects. *M.* is the abbreviation for *Make*. *D.* is the abbreviation for *Design*. *C.* is the abbreviation for *Collaboration*. *L.* is the abbreviation for *Learn*.

1.2.3 Challenges in Art & Craft Design-Aid Systems

In addition to requiring additional learning and training, digital design-aid tools impose different types of challenges depending on their design-aid strategies. **When using generic craft design tools to make physical craft designs, the lack of domain-specific support can bring major challenges for designers**. Because generic tools are not specifically made for a particular craft, craft designers might need to find workarounds to accommodate their design needs. For example, crochet designers who design with *Adobe Illustrator* might need to draft single stitch symbols before drafting a full pattern. Papercut designers need to check if all parts of the design are connected manually.

In some instances, designers can quickly identify these workarounds (e.g., find existing plugins or tutorials). It is worth noting that some generic tools are extendable. When encountering design tasks unsupported by the design software, users might have mechanisms to develop plugins or add-ons to assist their designing processes. In addition, tools like *Blender* also let their users control through scripts, which offers users more flexibility. In other cases, the limitation of the design tool can force designers to compromise their original design directions.

Most generic design tools fail to provide targeted simulation methods because of the lack of connection to specific craft domains. For certain craft domains, especially those not designed through 3D modeling methods, the design tool does not provide a default simulation function. Therefore, designers need to imagine the output or use other means to create simulations of the design. For instance, embroidery designers can use graphics editing software to draft their patterns. However, simulating

thread texture is not a commonly available function within graphics editing tools. For texture-rich embroidery styles, having realistic simulations could be potentially helpful. Nevertheless, designers either need to proceed with the execution process without simulating the design or have to produce the simulation using other means.

The experience of using generic design-aid software for specific craft design tasks could be similar to a supermarket shopping experience: the chef has all the ingredients they would possibly need, but they have to figure out the exact procedure by themselves. If the dish (i.e., design task) they plan to make is popular, they might find recipes (i.e., tutorials) to guide their creative processes. Otherwise, they need to experiment and explore on their own.

Instead of using generic design-aid systems, designers might also have access to domain-specific software for some craft disciplines. For example, *Pepakura Designer*[9] targets paper model designing tasks and helps designers to unfold 3D patterns. *StitchFiddle*[10] specializes in grid-based yarn pattern design. Although craft-specific design tools support designers with domain knowledge, this set of knowledge is often implicit. The implicit domain knowledge embedded in the tool makes it difficult to identify a particular tool's limitation. For example, *Pepakura Designer* approximates curve lines with straight lines before unfolding a 3D model. Without this knowledge, designers might have created 3D models with excessive curves, resulting in unmanageable patterns. Ideally, the domain information used in building a tool should be fully transparent to its users to assess the suitability between the tool and the design task.

[9] `https://tamasoft.co.jp/pepakura-en/`
[10] `https://www.stitchfiddle.com/en/chart/create`

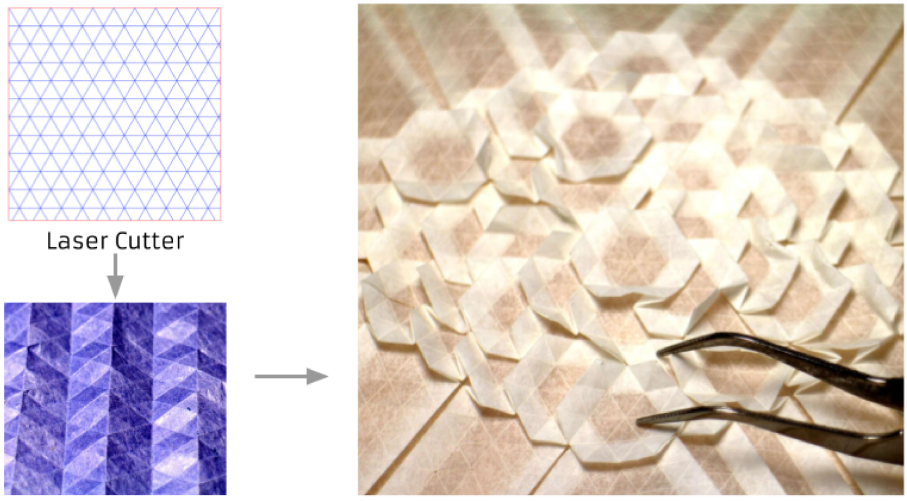

Figure 1.4: An origami tessellation example.

A more critical issue associated with many domain-specific tools is the lack of flexibility. Domain-specific tools often offer viable and sophisticated solutions to these tasks because they are designed to support a set of design tasks. Nevertheless, when designers need to perform a relevant task that is not designed in the original system, domain-specific tools reviewed in Section 1.2.2 often offer none or minimal support, therefore, they require users to make workarounds. For instance, producing fabrication-ready designs is another challenge that designers need to face when using small and craft-specific tools. Designers are often limited by supported exporting formats, machine-specific syntaxes, or even the design-aid tools' operation platforms. To illustrate, origami tessellation is the art of folding repetitive structures. By creating unit components repetitively, origami tessellation artists transform 2D patterns into 3D structures. Figure 1.4 displays an origami tessellation example.

Tess [11] is a Windows-only software for designing origami tessellation. While offering impeccable pattern-generating functions and a useful 3D simulation module, *Tess* lacks flexibility in its exporting format (only exports to pdf and png). Tessellation artists who intend to use machines to score the creases will need additional software to edit the design. It is possible and expected that tool developers could not foresee users' needs. For example, when *Tess* was created, making origami creases with a machine might not be a popular method. However, a craft design tool should be able to generate fabrication-ready designs in various formats.

Compared to the supermarket analogy for generic design-aid systems, domain-specific design-aid tools often provide a restaurant-like experience. These tools have "fixed menus" that offer reasonable solutions in many cases. But if users want to go off the menu, they need to seek other solutions, such as using additional tools for post-editing (e.g., using a file format converter).

In summary, while existing design-aid tools become increasingly powerful and intelligent, craft designers still face various challenges. At a high level, generic tools often lack domain-specific support. As a result, designers need to acquire craft domain knowledge and design tool knowledge independently. Although domain-specific tools can offer targeted support for domain-specific tasks, they often provide limited flexibility.

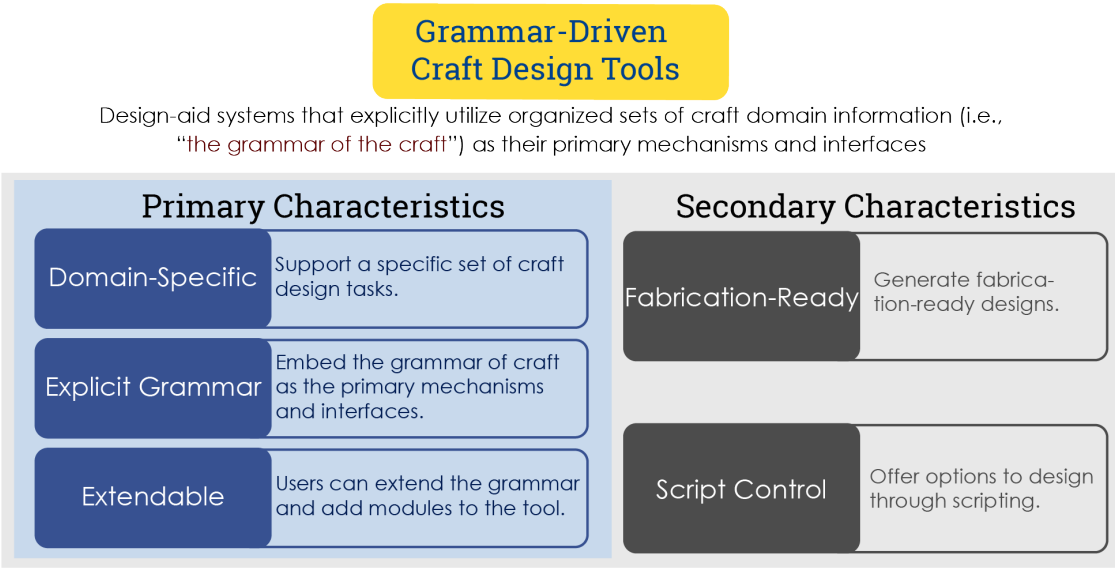

Figure 1.5: Overview of Grammar-driven Craft Design Tools

1.3 Grammar-Driven Craft Design Tools

Intending to tackle existing challenges associated with craft design-aid systems, I examine a new direction for making craft design-aid tools. I refer to this direction as the Grammar-driven Craft Design Tool (GCDT) Framework. Also, I refer to systems that adopt this framework as Grammar-driven Craft Design Tools (GCDTs). This section provides an overview of this framework by presenting the key characteristics that GCDTs have. Figure 1.5 provides an overview of these characteristics.

1.3.1 Primary Characteristics

Domain-Specific

GCDTs are craft-specific tools that support defined sets of design tasks (i.e.,

[11] `http://www.papermosaics.co.uk/software.html`

Domain-Specific). Instead of supporting a wide range of design tasks like generic design-aid tools (e.g., modeling software), GCDTs focus on solving tasks within a scope bound by the craft discipline. A core motivation for adopting this system-building strategy is that domain-specific tools offer more potential to bridge the gap between craft domain knowledge and design tool knowledge.

As discussed in Section 1.2.3, generic tools are robust design-aid systems, yet they are detached from specific design tasks because they do not have domain-specific information. Because of the lack of domain-specific information, designers also need to independently acquire domain knowledge and tool knowledge. For example, to design a crochet pattern using a generic vector graphics editing software, a designer needs to 1) learn icons associated with different stitches and 2) learn to use the editing software. In this case, these two processes are independent of each other. If the designer were to use software designed for crochet pattern drafting, the software might offer a library of stitch icons. While learning the meaning behind these icons, the designer is also familiarizing themselves with the design tool. By focusing on a specific craft discipline, GCDTs can offer targeted support and bridge the gap.

Explicit Grammar

Like other domain-specific craft design tools, GCDTs utilize and embed domain-specific information. For example, a crochet pattern design software contains crochet-specific knowledge, such as stitch terminology and pattern restriction. Nevertheless, users might not have a clear access to this set of knowledge, nor do they always offer guidance on utilizing this set of information. How to collect, structure, and embed this set of domain-specific knowledge into design-aid tools is entirely up to

the tool designers. In comparison, the GCDTs framework imposes two specific constraints on tool designers so that users of GCDTs have clear access to domain-specific information.

To start, GCDTs utilize an organized set of craft domain information that I refer to as the grammar of the craft. The grammar of the craft is an essential set of domain knowledge that constructs a language for describing and making designs in a craft domain. While grammars across craft disciplines differ from each other in terms of content, they share similar structures. Namely, each set of grammars contains three major parts: 1) Programmable Structure, 2) Creation Pipeline, and 3) Domain-specific Methods. In Section 1.4, I elaborate on the concept of grammar, as well as the process for extracting a set of grammar.

Grammar serves as GCDTs' primary mechanism and interface. Chapter II offers examples and a more detailed discussion about how this is achieved. With the combination of these two constraints, GCDTs connect domain-specific knowledge and design-aid tool. They also offer transparency over how these connections are built.

Extendable

As discussed in Section 1.2.3, domain-specific design-aid systems often offer limited flexibility because they are designed to support a specific set of design tasks. GCDTs address this issue by giving users options to add additional modules. If the current set of grammar cannot describe the design tasks that users have, they can add additional grammar and corresponding modules to GCDTs. The grammar-driven structure ensures that users can add additional rules as long as the additions

do not conflict with the existing set of grammar.

1.3.2 Secondary Characteristics

Besides these three primary characteristics, two secondary characteristics are associated with GCDTs that I developed in this dissertation. While "generating fabrication-ready design" and "offering control through scripting" are less critical than the grammar-driven structure, these two characteristics have the potential to improve users' experience and enable a broader range of creative strategies.

Fabrication-Ready

The outputs of GCDTs should be fabrication-ready. "Fabrication-ready design" has different meanings in different craft domains. For artifacts meant to be made with fabricators such as 3D printers and laser cutters, fabrication-ready designs indicate that the output format should be compatible with the associated fabricators. For artifacts that require manual construction (e.g., crochet and cross stitch), design-aid tools should consider how to support artisans' manual making process.

Assisting the making of fabrication-ready design means that systems developers will consider the physical making process of the final artifacts and provide support for the procedure. The specific strategy, approach, and implementations that system designers take to fulfill this requirement can differ across domains and tools. Chapter IV provides an detailed example on this topic.

Script Control

Offering script-based control is another characteristic of GCDTs that I describe in this dissertation. Including scripting support could lead to two major benefits.

It ensures that users can modify and extend the tool. As discussed in Section 1.3, GCDTs build upon sets of editable grammar. Allowing users to add customized scripts provides access to the grammar and gives users more control over their design outputs. For instance, users unsatisfied with certain output formats could add additional format-adjusting scripts to the tool.

Besides, including programming supports opens up opportunities for generative design. Generative design can have different meanings in different domains. In this dissertation, I refer to the generative design method as using algorithms to produce design outputs. In comparison, I consider parametric design (as described in *Aish and Woodbury* (2005)) a particular case of generative design where constraints are used to create designs. Other names, such as algorithmic design (e.g., *Jacobs* (2013)), share similar ideas. *Jacobs* (2013) provides a comprehensive review of challenges and advantages in incorporating a generative design approach in craft design domains. For instance, *Jacobs* (2013) argues that the generative design approach can produce precise and complex design variations. Yet, this approach adds additional challenges for designers to specify the constraints of the design task.

Generative designs are often tied to programming because major generative art tools supports generative design through scripting. Processing[12] and OpenFrameworks[13] are popular programming language/toolkit developed for creative coding. Other tools such as Grasshopper 3D [14] utilize block-like programming environment. Unlike tools such as *Jacobs* (2017, 2013); *Jacobs et al.* (2018) that focus on support

[12] `https://processing.org/`
[13] `https://openframeworks.cc/`
[14] `https://www.grasshopper3d.com/`

novice programmers, GCDTs can support programmers of different levels.

1.3.3 Suitable Craft Domains

While the GCDTs could be suitable for a wide range of craft, craft domains with the following characteristics might benefit more from the GCDT framework:

- Domains that have well-established rules might be more suitable than domains that have flexible guidelines. It is challenging to develop a comprehensive grammar for extremely flexible domains such as fluid painting[15] or general embroidery. In comparison, summarizing grammar sets for domains like paper marbling or circular crochet is easier.

- Domains that require precise control and complex design procedures might benefit more from the GCDT framework than domains with simple design procedures and fewer requirements for precision. For instance, bobbin lace [16] design might be more suitable for the GCDT framework than freeform collaging.

- Domains that benefit from the generative design method could be more suitable than domains that do not need scripting support. For example, cyanotype art, which is a photographic process to produce cyan-blue print, might not be an ideal domain for the GCDT framework.

GCDTs described in this dissertation have all primary and secondary characteristics. They are Python-based toolkits that require users to have expertise in

[15]Fluid painting is a painting method that produces images by pouring liquid paint on surfaces.
[16]Bobbin lace is the traditional way of making lace. It utilizes a tool called a bobbin to make intricate knots.

programming. I choose to implement script-based control for all three GCDTs to support the generative design approach. While the GCDT framework can potentially support a wider range of users, GCDTs in this dissertation are designed to support users who have expertise in creative programming. My GCDTs assist these users in their creative experiments that involve physical artifacts building. Because the GCDT structure is closely aligned with common concepts in object-oriented programming, these users would find the grammar structure quite familiar.

1.4 Domain Knowledge and Grammar of Craft

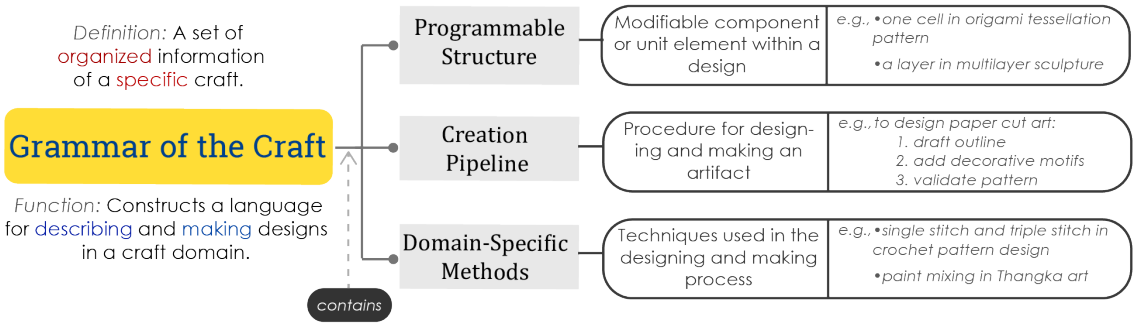

Figure 1.6: Grammar of the craft is an organized set of information extracted from craft domain knowledge. After a translate/refine process, the grammar of a craft consists of three major parts: 1) programmable structure, 2) creation pipeline, and 3) domain-specific methods.

The usage of grammar is a defining characteristic of GCDTs. The grammar of a craft is a set of organized information of a specific craft. There are two critical questions that a system developer might ask when they are considering developing a GCDT. First, what kind of information should they collect? Then, how to process the collected information?

In this section, I explain the concept of grammar by discussing the motivation behind this concept (Section 1.4.1). Then I discuss the composition of a typical set of grammar (Section 1.4.2). Figure 1.6 provides an overview of the concept "grammar."

1.4.1 Craft Domain Knowledge

To produce a design in a craft discipline, artisans need to gather relevant information and skills within that craft domain. For example, jewelry designers need to be familiar with various material; knitters need to understand knitting patterns. The grammar of a craft is essentially an organized version of this set of information. While each type of craft has its unique collection of domain information, I roughly categorize it into six categories.

- Goals & Constraints: Criteria to determine whether a design is satisfactory. Some craft has clear design goals, whereas others have flexible goals. For example, a functional mug design needs to hold a certain amount of liquid. In this case, the design goal is a constraint that designers choose to follow. In comparison, the design of a ceramic sculpture might have a much more complex and exploratory set of goals and constraints.

- Terminogy: Languages used when constructing and describing a craft. The set of information can include any term that has dedicated meaning in a craft. For example, *"single stitch"* in crochet, *"sugar lift"* in printmaking, and *"rocking"* in mezzotint.

- Material: Information related to the physical material used in making processes. E.g., knitwear designers need to differentiate wool yarn from acrylic yarn. With this group of information, craft designers should be able to answer questions such as
  - What kind of material are suitable for this craft?
  - What are the physical properties required?
  - How does the material impact the final output?
- Techniques: Actions that artisans perform during the designing and making processes. For example, crochet artists need to know the motion of the crochet hook when constructing a single stitch or a double stitch. In addition to individual techniques, this group of information also contains procedural knowledge related to the craft. For example, crochet artists need to know the procedure for finishing a piece.
- Common Practices: Information related to the typical and classic designs. In many craft disciplines, the design of artifacts has established patterns and motifs. Some of these common practices might be associated with the characteristics of the craft. For example, cross-stitch artisans sometimes do not secure the end of their threads because they might glue their embroidery pieces during framing processes. In some craft disciplines, these common practices might have cultural and historical meanings. For instance, there are an extensive set of common traditional motifs in Chinese paper cut art.

1.4.2 Turning Domain Knowledge into Grammar

The process and strategy for obtaining domain knowledge differ across craft disciplines. Domain knowledge is scattered and unorganized information for many craft disciplines. There are many potential sources to acquire this knowledge. For example, artisans can learn from instructional books such as *Leaf* (1984) for intaglio printmaking, *Chamberlin and Corbet* (2017) for goldwork, and *Birmingham* (2010) for pop up structure design.

Besides books, artisans can obtain craft-related information from articles, online forums, and video tutorials. For instance, *Ravelry*[17] is an online platform for thread artists to discuss and share knitting/crochet patterns. In general forums such as *Reddit*[18], there are also dedicated sections where artisans share information regarding a specific craft. Artists can learn from video content. There are many tutorial videos that store craft domain knowledge. Artisans also have many options to access online classes from services such as *Skillshare*[19] besides buying these tutorial DVD sets such as *Ross* (2017).

Comparing to texts/video content, workshops and in-person classes offer artisans more hands-on opportunities to obtain craft domain knowledge. In some craft disciplines, in-person teaching is the primary method for passing on craft-related information for many reasons. For instance, metal jewelry artists might find it difficult to only learn from books and videos because of equipment training. Cultural

[17] `https://www.ravelry.com/`

[18] E.g., `https://www.reddit.com/r/printmaking`

[19] `https://www.skillshare.com/` is an online learning community where content creators can share project-focused instructional videos.

influence and traditions might also impact how the craft knowledge is preserved and accessed. For example, Thangka artists often learn from other painters through apprentice training (*Gamble* (2001)).

After collecting domain knowledge, design-aid system designers face a more critical challenge: how to process this knowledge? In Section 1.4.1, I discussed five categories (i.e., goals, terminology, material, techniques, and common practices) of domain knowledge. Nevertheless, craft information that is organized into these five categories still cannot be directly used in developing a design-aid tool because each category of information contains mixed levels of complexity, abstraction, and priority. For instance, "a chain stitch must connect to two stitches unless it is the beginning stitch or ending stitch of a pattern" is a specific constraint in crochet design. Without any strategy for organizing and hosting information, it is unclear how tool designers should embed this information into their tools.

To provide tool designers more guidance over the information organizing stage, I use the concept "grammar" as a structure to host information. The concept of "Grammar of the Craft" is inspired by the grammar of graphics in information visualization. The Grammar of Graphics is a foundational framework in information visualization. It consists of graphical attributes, such as position and hue, that designers can use to encode data (*Wilkinson* (2012); *Munzner* (2014)).

The Grammar of Graphics constructs a language for describing and designing information visualizations. In a visualization authoring process, designers provide inputs from three aspects: 1) data, 2) a set of encoding rules (following the grammar of graphics) 3) additional specifications that adjust non-data-related attributes.

Mainstream visualization authoring tools such as *Tableau*[20], *D3.js*[21], *Vega*[22], and *Matplotlib*[23], all embed the Grammar of Graphics in their designs because visualizations are "written" with visual languages that follow the grammar (*Bostock et al.* (2011); *Satyanarayan et al.* (2016)).

Many collections of craft knowledge are presented as grammar-like specifications. Previous studies display the possibilities of extracting grammars within different domains such as wire-binding figures (*Noel* (2015)), Celtic knots (*Cromwell* (2008)), and bobbin lace (*Irvine and Ruskey* (2014)). Nevertheless, tool developers might still find it difficult to directly adopt these grammar specifications because these sets of grammars take various forms and are not designed for tool building.

To solve this issue, I identified structures that a set of grammar should have. A craft's grammar has three parts: programmable structure, creation pipeline, and domain-specific methods. In this dissertation, I examined three unrelated craft domains and experimented with various grammar structures. Through these experiments, I summarize this three-part structure (see Figure 1.6).

Programmable Structure

Programmable structures are components or unit elements within a craft design. These structures can have physical associations. For example, a sheet of material is a unit component in multilayer sculpture. In digital systems, this sheet of material can be represented using various forms, such as a blank canvas in graphics-based editing tools or a class in a programming environment. The term "programmable"

[20] `https://www.tableau.com/`
[21] `https://d3js.org/`
[22] `https://vega.github.io/vega/`
[23] `https://matplotlib.org/`

is used here to indicate that designers can edit the properties of these structures. In origami tessellation, each cell within the pattern is a programmable structure. By modifying the shape and location of individual cells, designers create different tessellation structures.

If a structure has associated constraints, these constraints become properties of this structure. Using the example of chain stitch in crochet again, I treat one chain stitch as a unit component. The constraint that "a chain stitch must connect to two stitches unless it is the beginning stitch or ending stitch of a pattern" can be organized into a validation method associated with this component. Whenever users create a chain stitch, this associated method can test whether the creation is valid.

Creation Pipeline

The second part of the information that a set of grammar should contain is the creation pipeline. The creation pipeline contains procedure information that users of design-aid systems need to follow to make a design. For example, to design a paper cut piece, a potential pipeline can include three steps: 1) draft the outline, 2) add decorative motifs, and 3) validate the pattern.

In real-world design scenarios, the creation process is likely to be non-linear. Studies such as *Hanington and Martin* (2012) and *Kumar* (2012) examine the design and design-thinking processes. They suggest that designers need to iterate over their designs. Similarly, the creation pipeline here can be non-linear. Specific steps might depend on previous steps (e.g., create a canvas before making any mark), yet users likely have some flexibility over the creation procedure.

Domain-specific Methods

Domain-specific methods are the last part of the grammar of craft. Each method represents a technique used in the designing and making process. These techniques can also take various forms in digital systems. For example, in origami pattern design, mountain fold and valley fold are two primary methods that all origami design software should have. In crochet pattern design, automatically arranging stitches into a circle could be a helpful technique.

With these three components, the grammar of a craft constructs a language for describing and making designs in this craft domain. GCDTs use grammar as their primary mechanisms and interfaces so that designers can "write" using the language constructed by the grammar. Chapter II provides a complete example for extracting and utilizing grammar in GCDTs.

1.5 Building GCDTs

In summary, GCDTs are design-aid systems that explicitly utilize organized sets of craft domain information (i.e., the grammar of the craft) as their primary mechanisms and interfaces. The design of the GCDT framework is motivated by existing design-aid tools and research in HCI. GCDTs have unique characteristics that are designed to tackle existing challenges associated with current craft design-aid systems. In chapters II, III, and IV, I elaborate on these characteristics and provide examples of GCDTs developed in three different craft domains.

First, *InfiniteLayer* is a design-aid tool that supports multilayer sculpture design (Chapter II) [24]. Multilayer sculpture is the art of creating overlaps. By processing

[24]This chapter is adopted from the manuscript titled "*InfiniteLayers: A Programming Toolkit for*

and layering sheets of material such as paper, wood, and fabric in specific ways, artists transform overlapping 2D designs into 3D forms. As computer-controlled fabricators become increasingly available, creating multilayer sculptures with machine assistance becomes viable and prevalent. However, creating fabrication-ready designs remains a challenging task, especially for intricate artwork with many layers. Also, existing tools often require artists to mentally translate between 2D and 3D forms, making rapid prototyping difficult. To address the software gap in multilayer sculpture creation, I present *InfiniteLayer*, a programming-based toolkit that supports the design of intricate and algorithm-driven multilayer sculptures. By synthesizing common techniques and constraints of this art form, *InfiniteLayer* supports the creation and manipulation of fabrication-ready designs. Furthermore, *InfiniteLayer* provides 3D simulation to assist rapid prototyping. I demonstrate the power of *InfiniteLayer* by showcasing a wide range of designs that it enables. Through this example, I document and discuss grammar extraction processes.

I present the development process of a GCDT in the domain of mark-making tool design (Chapter III)[25]. Mark-making tools enable artists to produce their imagined art in various forms. Despite the considerable variations of nibs, brush, stamp, and marker designs, artists continue developing unique mark-making instruments. To create ideal and unique marks with different mediums, artists modify their tools (e.g., cut brushes to specific shapes) or find alternatives (e.g., use toothbrushes and sponges). The availability of fabrication technologies enables a broad new class of

Multilayer Sculpture Design," which is co-authored by Shiqing He and Eytan Adar.

[25]This chapter is adopted from the manuscript titled "*Inventing Creative Mark-making Tools*," which is co-authored by Shiqing He and Eytan Adar.

mark-making tools. However, designing and fabricating a mark-making instrument requires many skills such as drafting, modeling, material handling, and manual assembly. As a result, creating unique and intricate mark-making tools remains challenging. This chapter examines existing design and fabrication processes for mark-making instruments such as pen nibs, brushes, and stamps. I identify opportunities and design space for personalized mark-making tools. After testing various fabrication methods such as 3D printing, casting, and manual construction, I contribute an open-source toolkit, *MarkMakerSquare*, that supports the design of creative mark-making tools. I demonstrate the range and limitations of the fabricated mark-making techniques. I also reflect on challenges encountered for developing design-aid systems that leverage multiple fabrication methods and material.

Lastly, Chapter IV presents *ThreadPlotter*, a GCDT that supports the design and fabrication of plotter-based delicate punch needle embroideries[26]. *Punch needle embroidery* is a unique type of embroidery that uses loops of threads to create designs. Technology for punch needle embroidery ranges from popular handheld manual tools to high-cost industrial tufting machines. Computer-controlled punch needle fabrication tools remain out-of-reach for most practitioners. This work describes how a low-cost X-Y plotter can be repurposed to support punch needle embroidery fabrication. By adding easy-to-make physical accessories coupled with a novel software toolkit, I support the production of delicate and precise punch needle embroideries with minimal manual labor. After examining and evaluating the potential and chal-

[26]This chapter is adopted from the manuscript titled "*Plotting with Thread: Fabricating Delicate Punch Needle Embroidery with X-Y Plotter*", which is co-authored by Shiqing He and Eytan Adar. It is published at DIS '20, July 6–10, 2020, Eindhoven, Netherlands (*He and Adar* (2020)).

lenges of converting X-Y plotters into punch needle embroidery fabricators, I propose a set of design and fabrication guidelines specific to plotter-based punch needle embroideries. I demonstrate how this novel fabrication approach enables the production of a wide range of artifacts and textures.

While these three GCDTs have all primary and secondary characteristics described in Section 1.3, each of them has a different focus. The development process for *InfiniteLayer* provides insights for grammar extraction. In comparison, *MarkMakerSquare* emphasizes the importance of extendability. Because plotter-based punch needle is an unconventional fabrication method, *ThreadPlotter* is an example of GCDT that produces fabrication-ready design. Through these tools, I examine how GCDTs' unique characteristics assist creative activities.

CHAPTER II

Multilayer Sculpture Design

2.1 Introduction

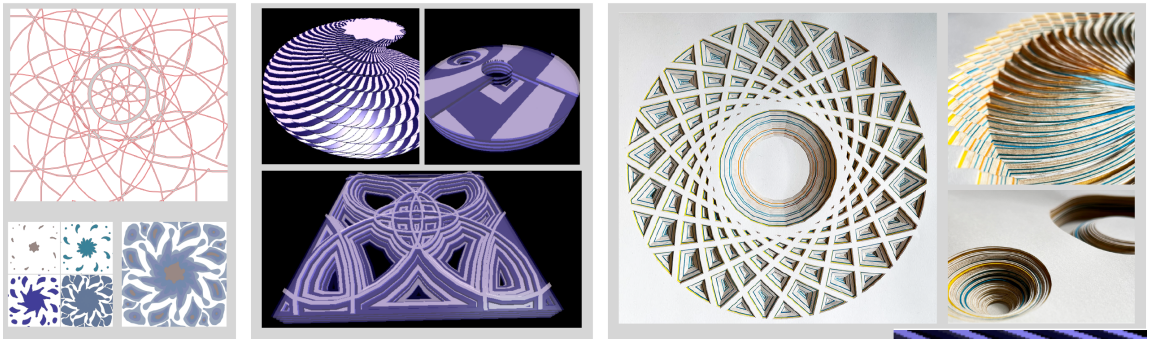

Figure 2.1: *InfiniteLayer* is a Python-based toolkit that assists multilayer sculpture design. It provides foundational structures and essential methods for designers to draft, import, and manipulate 2D designs (LEFT). It aids rapid prototyping by delivering 3D simulations without requiring modeling expertise. Also, it offers flexible controls over rendering settings/formats to produce designs that are compatible with a variety of fabricators (RIGHT: artwork fabricated with a laser cutter).

The usage of grammar is a defining characteristic of GCDTs. In this chapter, I discuss the grammar extraction processes through a GCDT developed for multi-

player sculpture design. Through this example, I present the process for collecting, refining, and utilizing domain-specific information in the development process of a GCDT. Multilayer sculpture is a category of 3D structures that consist of layers of material. By manipulating individual layers, designers can create artifacts using various material such as paper, wood, fabric, and acrylic sheets. Besides creating aesthetically pleasing structures for art and decoration purposes (e.g., paper cut by *Kubo* (2009)), installation art (e.g., *Schama* (2019)), designers also make multilayer structures as intermediate tools for other creations (e.g., stencils for painting and screen printing (e.g., *Jain et al.* (2015); *Igarashi and Igarashi* (2010)). In this chapter, I first present the domain-specific information collected for multilayer sculpture in section 2.2. In section 2.3, I produce a set of grammar by organizing the information into three major components: programmable structures, creation pipeline, and domain-specific methods.

Paper-based layer art is a common form of multilayer sculpture with a rich history across many cultures. Artisans around the world have been using various hand-held tools to produce delicate designs (*Ryan and Avella* (2011)). Manually making multilayer sculptures could be a meditative process, but it also comes with several limitations. Manual fabrication could be a labor-intensive task that requires extensive technical training. One of the most challenging aspects of manually making is the limited range of material that one can precisely cut using hand-held tools. Artisans can easily find tools to cut delicate designs on paper, but cutting the same design on more rigid material such as acrylic sheets or metal by hand is considerably more challenging. Likewise, extra delicate material such as tracing paper are easily tearable

during the manual cutting process. As computer-controlled fabricators, such as laser cutters, CNC mills, vinyl cutters, and die cut machines, become increasingly available, using machines to assist the making of multilayer sculptures has become more prevalent. Besides offering precise control over how a design is cut, fabricators also open up the range of material that designers can easily manipulate. Additionally, designing and iterating digitally before committing to a design can save significant prototyping time and material cost.

When creating these 3D forms digitally, it is tempting to search for 3D modeling tools. While slicing a 3D model into layers is a viable design approach for some tasks, it is unsuitable for all multilayer sculpture design tasks. The fundamental issue comes from how the digital design process connects to the physical creation process. When building a 3D model, designers typically take an additive/subtractive approach, just like how they would shape a ball of clay or chip away a corner of the stone. This approach is powerful for designing connected and solid objects. For example, users can build a dinosaur model, slice it into layers, and then assemble layers back into a solid multilayer dinosaur sculpture.

Nevertheless, specific multilayer design tasks require designers to focus on one slice of material at a time. While shaping a ball of clay into an interlacing structure (e.g., a simple plain weave) is challenging, building the same structure with clay slabs is considerably more manageable. Similarly, creating a 3D model for multilayer sculptures with interlacing structures is counter-intuitive. It requires designers to think in 2D: they need to shape their digital "modeling clay" into fixed, flat sheets before modeling.

Because the primary material for multilayer sculpture is flat, it calls for a design process similar to how artisans play with a stack of paper. By going through the stack layer by layer, designers can take advantage of this layering process and design overlapping shapes that are difficult to imagine otherwise. Many commonly used techniques in multilayer sculptures, such as creating partial overlap, combining different material, and intentionally covering previous layers (e.g., for lightbox design), are easier to design using a 2D design approach. Similarly, when designing other 3D objects intended to be fabricated with sheets of material, designers might opt for 2D design tools such as *LaserOrigami* (*Mueller et al.* (2013)) and *CutCAD* (*Heller et al.* (2018)), because "the underlying 2D design principles are easier to understand (*Heller et al.* (2018))."

Therefore, many multilayer sculpture designers rely on 2D vector graph authoring tools, such as *Adobe Illustrator* and *Inkscape*, to draft 2D designs. Additionally, there are specialized software designed for making painting stencils (e.g., *Jain et al.* (2015); *Igarashi and Igarashi* (2010)), paper cut art (e.g., *Liu et al.* (2018)), and iris folding artwork (e.g., *Igarashi et al.* (2016b)). Designers can still find their tasks challenging to tackle even with the assistance of existing tools. To start, when designing highly complex and precise pieces, designers need to repeatedly perform basic actions, such as aligning, resizing, and translating. Also, while systems developed specifically for paper cuts design and stencil-making can help users vectorize raster images and ensure that all parts are connected, they often offer minimal flexibility. For example, using tools focused on extracting and processing images (e.g., *Jain et al.* (2015); *Igarashi and Igarashi* (2010); *Meng et al.* (2010); *Liu et al.* (2018)), users need to

rely on additional tools for post-processing, such as adjusting file format and changing line thickness/color. Most domain-specific tools (e.g., *Igarashi and Igarashi* (2010); *Meng et al.* (2010); *Liu et al.* (2018); *Higashi and Kanai* (2016); *Yang et al.* (2019); *Xu et al.* (2007); *Liu et al.* (2020)) also focus on supporting single-layer designs and have limited assistance for creating multiple layers at once.

The lack of prototyping support is another challenge associated with current design-aid tools in this domain. Although a layer-by-layer design approach is more viable for many tasks, it does not suggest that only viewing 2D graphs is sufficient for all design tasks. To check whether patterns on each layer "contribute" to the desired 3D form, designers need to see how layers stack together. Currently, artists need to rely on additional 3D modeling tools to visualize their designs accurately. Manually constructing these 3D models is a labor-intensive task, especially when the design contains many layers. Alternatively, designers can approximate the 3D layering effect by digitally laying 2D designs together. For example, by showing the residual ("onion skin") of the previous layer, designers can imagine how two layers overlap to create a 3D form. Figure 2.1 (bottom left) shows an example where the designer creates four layers and digitally stack them together to visualize the finished artwork.

To address these issues, I designed *InfiniteLayer*, a Python-based programming toolkit that assists the design and fabrication of multilayer sculptures. While *InfiniteLayer* supports design tasks with different complexities, it is most suitable for intricate designs that require precise controls. Also, while *InfiniteLayer* does not limit the design approach that users can take, it is designed to support the making

of algorithmic craft, which takes advantage of the parametric and generative design approach. By examining representative artwork in this domain, I summarize foundational structures and core techniques in this art form. I further extract and refine this fundamental domain knowledge and transform it into the infrastructure and methods that *InfiniteLayer* provides. Using *InfiniteLayer*, designers can create intricate artwork with several lines of code. In addition to supporting the creation of elaborated and algorithm-driven designs, *InfiniteLayer* also provides a 3D simulation module that assists users' prototyping processes. Users can inspect and interact with the web-based 3D models without having to construct these models manually.

In this chapter, I introduce a GCDT named as *InfiniteLayer*. It is a programming-based solution to tackle multilayer sculpture design challenges. I develop and evaluate *InfiniteLayer*, an open-source tool that supports the design and simulation of multilayer sculptures using programming. This tool offers essential infrastructure and methods that summarize this art form's core techniques and constraints. I display various approaches that users can take when using the tool. I also present a collection of multilayer sculptures designed using *InfiniteLayer*. Through this example, I demonstrate the grammar extraction process of a GCDT.

2.2 Related Work

In this section, I review artwork, tools, and fabrication methods related to multilayer sculpture design. By examining various forms of multilayer sculpture, current design-aid tools, and the design/making methods associated with this art form, I explain motivations for building the *InfiniteLayer*.

2.2.1 Multilayer Sculpture and Related Art

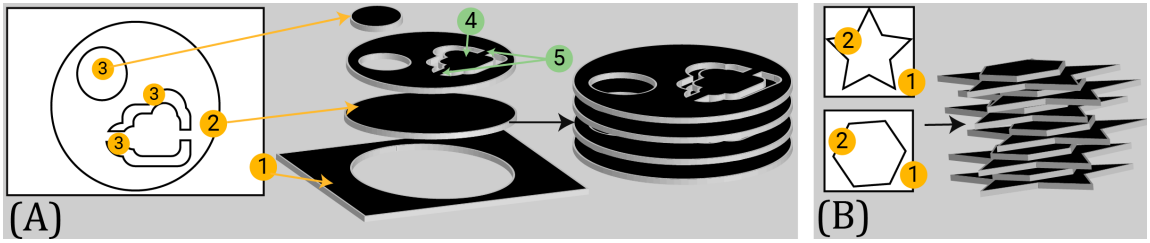

Figure 2.2: I describe components of multilayer sculpture using the term "layer (a flat a sheet of material)" and "stencil (shapes to be cut)." Design A illustrates an example that utilize three types of stencils: a *base* stencil (1), a *boundary* stencil (2), and multiple *pattern* stencils (3). Certain types of stencils can also be described as (4)"island shapes", and (5)"bridges" that connecting island shapes to the main structure. Design B provides an example where each layer only contains *base* stencil and *boundary* stencil.

To describe and analyze multilayer sculpture, I first clarify the terms I use to describe this art (Figure 2.2). A "layer" denotes one flat sheet of material. The layer is the *unit* component of a multilayer sculpture design. In a digital design setting, a layer can be an infinitely large canvas/plane. Practically, the dimensions of the layer are constrained by any physical material that designers would like to produce. I use the term "stencil" to represent a group of shapes to be cut. This definition is inspired by physical die-cut stencils that are used to cut out shapes from paper or metal sheets. To create a multilayer sculpture, artisans need to use different types of stencils. For example, when creating paper cut art, an artist first trims a postcard-sized sheet of paper from a large paper roll. The postcard-size trimming template is an example of a **base** stencil that represents the shape of the unprocessed material. Then, the artist cuts out a smaller circle that serves as the main structure of the

paper cut. The template used for cutting the circle is a *boundary* stencil. Next, the artist cuts other shapes away from the circle to create the paper cut design, and I refer to these shapes as *pattern* stencils.

Figure 2.2 provides a visual example of my definitions. Each layer of the sculpture can contain one or more stencils. Also, each layer can have different boundary stencils. I note that my definition of stencils differs slightly from painting stencils, which primarily consider stencils as a sheet of material with cut-away patterns (*Jain et al.* (2015)). In comparison, my definition has a closer connection to die-cut stencils, which are intermediate tools instead of the final result.

To understand components, techniques and constraints of this art form, I reviewed a collection of representative multilayer artwork from notable artists such as Maud Vantours, Charles Clary, Gabriel Schama, Eric Standley, Martin Tomsky, Julia Ibbini, and Stephane Noyer. These artists have different styles and create for a broad set of presentation goals. Multilayer sculptures can be installation pieces (e.g., *Ibbini and Noyer* (2021); *Standley* (2020); *Clary* (2013)), graphics design components (e.g., *Vantours* (2020)), and even jewelry (e.g., *Tomsky* (2019)). Furthermore, the subject matter of these creations varies dramatically. For example, artists used motifs inspired by nature (e.g., *Vantours* (2020)), figures (e.g., *Tomsky* (2020)), architectural elements (e.g., *Standley* (2020)), and abstract/geometrical shapes (e.g., *Vantours* (2019); *Clary* (2013); *Schama* (2019); *Ibbini and Noyer* (2021)). In addition to the subject matter, artwork also vary by their material. Different types of papers such as card stock, construction paper, and specialty papers create different color and textures (e.g., *Vantours* (2019, 2020); *Clary* (2013); *Standley* (2020); *Ibbini and Noyer*

(2021)). Wood is another popular choice that provides a large selection of texture and finishes (e.g., *Schama* (2019); *Tomsky* (2020, 2019)).

Apart from multilayer sculptures, I also examined several related crafts. Single-layer stencils have a wide range of applications. Besides serving as art and decorations, they are used as intermediate tools for painting (e.g., *Jain et al.* (2015)) and screen printing (e.g., *Griffiths* (1996)). Paper Cutting is a broader art category that often overlaps with multilayer sculpture. *Ryan and Avella* (2011) displays a variety of techniques that are frequently shared with multilayer sculptures design.

Paper cut art has different characteristics associated with the creators' design context and culture. It is a versatile form of art with a long history. For example, Chinese paper cut (剪纸 jiǎnzhǐ) is a folk art that focuses on creating two-tone images using traditional motifs (e.g., *Meng et al.* (2010); *Liu et al.* (2020, 2018)). In addition to cutting from flat material, jiǎnzhǐ also uses paper folding techniques to create symmetrical designs (e.g., *Liu et al.* (2005, 2018)). In Japan, paper cut (Kirié, 切り絵) artists such as Kubo Shu (久保 修) focus on creating painting-like multilayer sculptures with washi paper (*Kubo* (2009)). Iris folding is a unique form of multilayer sculpture. Instead of creating multiple layers that share similar boundary shapes, iris folding pieces include a base stencil, a top stencil with at least one pattern (the "iris" or "aperture"), and many paper strips. By layering these paper strips in specific order and location, artists form "a spiral pattern behind an aperture (*Igarashi et al.* (2016b))."

I designed the *InfiniteLayer* to support a wide range of layered structures by focusing on providing foundational infrastructure instead of reinforcing specific styles.

As a result, designers can create sculptures with different motifs, purposes, and levels of complexities. For example, while the *InfiniteLayer* does not automatically analyze and validate symmetrical design like *Liu et al.* (2005), users can make symmetrical stencils. Similarly, although the *InfiniteLayer* cannot automatically create section suggestions for Iris rotation similar to *Igarashi et al.* (2016b), users can make designs in the iris folding style.

2.2.2 Design-Aid Tools for Multilayer Sculpture and Related Craft

Artists currently have three major types of design-aid tools to create multilayer sculptures and related crafts using a 2D design approach. First, artists can use generic vector graph editing tools such as *Adobe Illustrator* and *Inkscape*. Despite the lack of domain-specific supports such as 3D simulation and batch adjustments for multiple layers, generic graphics manipulation tools are popular for multilayer sculpture design because they provide an extensive range of powerful shape manipulation functions. Also, they are likely to offer exporting formats that are compatible with various fabricators, such as laser cutters and CNC mills.

In addition to generic software, design and research communities have developed several systems to support the design of stencils, paper cuts, and iris folding pieces. Some systems focus on extracting and optimizing stencil and paper cut-ready patterns from raster images. There are a number of techniques and algorithms for this type of conversion (e.g., *Xu et al.* (2007); *Meng et al.* (2010); *Liu et al.* (2020)). Other approaches provide design support for paper-cut patterns that involves folding (*Liu et al.* (2005, 2018)). Besides paper-cut related tools, there are algorithms and soft-

ware for generating single-layer stencils for paintings (e.g., *Bronson et al.* (2008); *Igarashi and Igarashi* (2010)). Specifically, *Holly* is a system with an interactive interface for drafting stencils (*Igarashi and Igarashi* (2010)). By processing individual stroke input from users, *Holly* generates valid painting stencils by building "bridges" that connect "island" shapes to the main structure (see Figure 2.2) for an illustrated example for islands and bridges). *Stencil Creator* is a system that automatically generates sets of multi-color cut-out templates (*Jain et al.* (2015)). It utilizes an algorithm that generates sophisticated multilayer stencil sets from images using a random field energy formulation. Furthermore, some tools focus on guiding designers through the designing and cutting processes. They provide design guides and step-by-step cutting instructions (e.g., *Liu et al.* (2018); *Higashi and Kanai* (2016)) or physical assistance tools for paper cut artists (*Higashi and Kanai* (2019)).

Some systems focus on designing 3D structures inspired by 2D stencil designs. For instance, it is possible to transform 2D stencil designs into 3D-printed relief sculptures (*Yang et al.* (2019); *Jung et al.* (2020)). Also, paper cut art can serve as a style reference for animated 3D structures (*Li et al.* (2007)). While this group of systems might not directly assist the building of multilayer sculpture, they demonstrate a wide range of potential applications for stencil-related art.

2.2.3 Fabrication and Procedural Generation

While existing systems such as *Igarashi and Igarashi* (2010); *Jain et al.* (2015) provide a solid foundation for creating stencils, there are two common limitations. First, they focus on generating layers by extracting patterns from raster images or

stroke inputs, therefore having limited supports for vector graph manipulation. Although supporting vector graph manipulation might be optional for manual construction, artists who use fabricators to implement their design would find it critical to have precise control over individual paths rendering settings. For example, although *Stencil Creator* (*Jain et al.* (2015)) provides raster-to-vector conversion automatically, users need to rely on external software to perform actions such as adjusting canvas size, updating stroke thickness, add/remove points on paths.

Second, to create complicated designs with many layers, existing systems would repeatedly require users to perform similar actions. For example, designers need to enlarge the circular pattern stencil and adjust these circles' locations repeatedly to create a simple sculpture with a circle enlarging at each layer. Actions such as "center," "move," "offset," and "union" will be performed hundreds or thousands of times for one project. When making designs that have clear procedures, scripting-based controls could help to increase the speed and precision. It also handles repetitions (*Jacobs* (2013)).

To address these two issues, I designed my system to be a vector-focused, scripting-based tool. *InfiniteLayer* gives users controls to every detail of their vector images to ensure that results created with *InfiniteLayer* are fabrication-ready. For instance, users can easily adjust settings such as size, stroke, color, thickness, and exporting format. In addition, although it primarily supports vector graph manipulation, *InfiniteLayer* provides utility functions for vectorizing raster images.

InfiniteLayer is a programming-based tool that supports algorithm-driven designs that are highly complex and precise. With the development of programming-based

visual-focused tools like *Processing* (*Reas and Fry* (2007)) and *P5.js* (*McCarthy et al.* (2015)), constructing 2D and 3D design through programming has become a compelling and viable design method (*Levin and Brain* (2021)). In addition to generic programming-based tools that support procedural generation, the design and research community also developed programming-based tools for specific art and crafts forms. For example, *Jacobs* (2017) and *Jacobs et al.* (2018) focus on supporting procedural generation in manual drawing and painting.

Because multilayer sculptures often require repeated actions and precise graphics manipulations, I believe that a programming-based tool could help designers quickly and precisely manipulate their designs using loops, conditions, and functions. While users can still manually adjust individual shapes, a programming-based toolkit opens up possibilities for procedural generation and parametric design.

2.3 Design Multilayer Sculptures with *InfiniteLayers*

I designed *InfiniteLayer* with several objectives. The most fundamental goal is to provide an infrastructure to hold and process information related to a design task. With such an infrastructure, the *InfiniteLayer* can automatically and effectively handle basic and repetitive common actions. For example, with the current vector graph editing tools (e.g., *Adobe Illustrator*) and programming-based design tools (e.g., *Processing*), multilayer sculpture designers need to create the canvas for individual layers manually. Ensuring that these canvases have the correct setups, such as width, height, and margins, is a tedious yet unavoidable step that these tools cannot help automate because they are designed for a broader set of design goals.

Because *InfiniteLayer* is explicitly designed for multilayer sculpture, it can automatically handle these low-level, repetitive actions. Moreover, the *InfiniteLayer* needs to assist in the generation and manipulation of vector graphs. Constructing a vector image from scratch requires diverse expertise in vector graphics (SVG), ranging from the composition of a <path> element to matrix transformation. The *InfiniteLayer* provides convenient methods that handle these essential actions so that users can focus on design.

Besides providing reliable infrastructure and convenient methods for designing layered structures, objectives such as "supporting diverse design approaches" and "assisting prototyping/fabrication" also impact the design of the *InfiniteLayer*. Therefore, this section presents the core creation workflow and methods that the *InfiniteLayer* supports and provides.

2.4 Creation Workflow

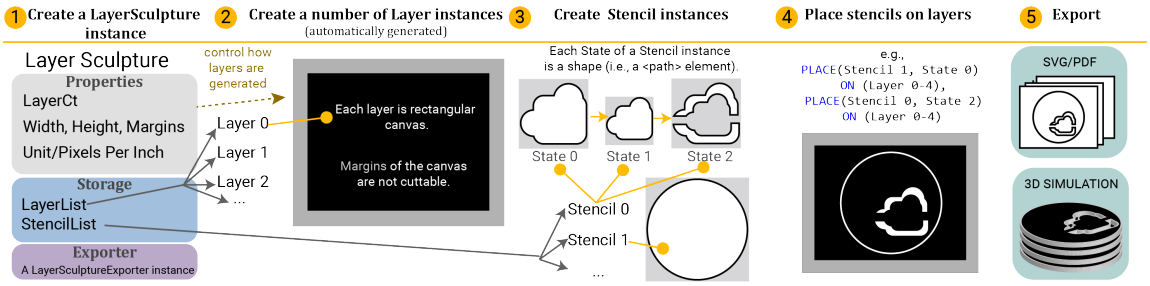

Figure 2.3: A creation workflow using *InfiniteLayer*.

To support multilayer sculpture design, *InfiniteLayer* offers four major classes: `LayerSculpture`, `Layer`, `Stencil`, and `LayerSculptureExporter`. `Lay-`

erSculpture is a primary class that hosts all information and methods related to a multilayer sculpture project. It processes users' inputs (stored in a dictionary) as basic settings for the project. Minimally, users need to provide two groups of information: 1) the number of layers in this sculpture (LayerCt), and 2) The size of each layer (e.g., the width, height, and margin settings of the base material). Users can adjust global settings such as the unit (default = inch) and pixels per inch (default = 96).

A LayerSculpture instance has storage for Layer instances and Stencil instances. A Layer instance represents a sheet of material to be cut. Essentially, it contains 1) a rectangular canvas that is defined by width, height, and margins (left, right, top, bottom), and 2) a list of pointers to shapes that will be cut from the canvas.

A Stencil instance contains various states of a group of shapes. In the physical layered structure design process, artists often need to manipulate their die-cut stencils (e.g., rotate the stencil, bend the cookie cutter) so the same stencil can create different cuts. I model this process by designing the Stencil as a storage class that tracks a series of shapes generated from one original shape. Each copy of the shape is called a State. Each State instances corresponds to a <path> element in a SVG file. The first shape (State 0) is the original shape, and all successor states are manipulated copies of the previous states. State instances are automatically indexed so that users can easily track their progression. Users can also add additional labels to States instances.

Besides holding global properties, layers, and stencils, a LayerSculpture in-

stance automatically initiates a `LayerSculptureExporter` instance to handle everything related to file export and simulation. Thus, users are likely to go through a five-step process for designing a multilayer sculpture using *InfiniteLayer* (Figure 2.3).

1. Users initiate a `LayerSculpture` instance and input required parameters such as `LayerCt`, `Width`, and `Height`.
2. The `LayerSculpture` instance automatically creates `Layer` instances according to the settings.
3. Users create `Stencil` instances and manipulate these instances to create new `States` of these stencils.
4. Users place `Stencil` on layers by specifying the stencil-state pairs and the layer's indexes that they will be placed on.
5. The `LayerSculptureExporter` exports the finished designs into SVG and/or PDF. It can also generate a WebGL-based 3D simulation.

2.5 Creating and Manipulating Stencils

By analyzing the list of representative artwork described in Section 2.2, I extracted an essential set of commonly used techniques in multilayer sculpture design. I refined these techniques into convenient methods of *InfiniteLayer*. In addition, I grouped these methods into four categories by their purposes. Figure 2.4 provides an overview of these methods.

A) Methods for Creating Stencils

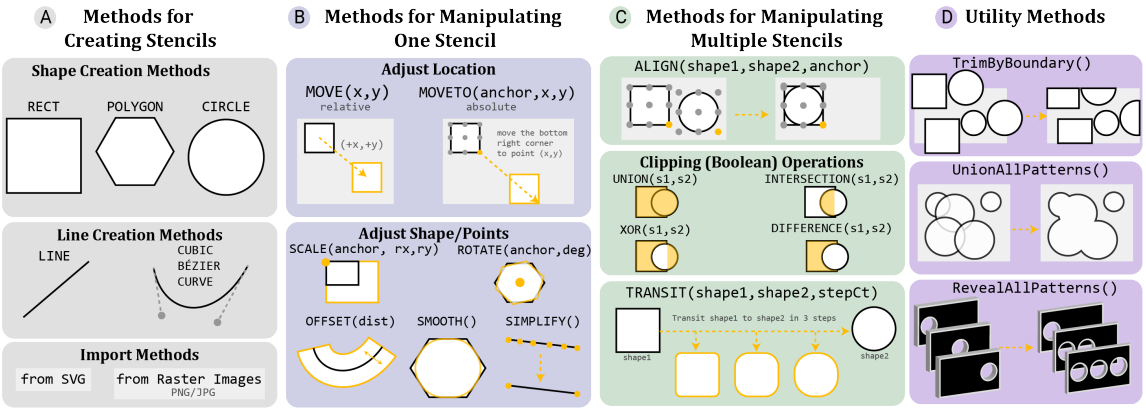

Figure 2.4: Overview of methods provided by *InfiniteLayer*.

InfiniteLayer provides a group of functions that initiate or import path information. This group of functions automatically creates `Stencil` instances to store this information. Users can create basic shapes and lines such as rectangles, polygons, straight lines, and curves. All path information needs to be processed into straight line segments because algorithms for self-intersection detection and boolean operations generally focus on lines and polygons. Therefore, when users create a circle, *InfiniteLayer* automatically approximates the circle by generating a regular polygon with a large number of sides. Similarly, *InfiniteLayer* provides functions to approximate bézier curves with straight lines.

In addition to creating stencils from scratch, users can import stencils from existing drawings. *InfiniteLayer* offers an importing/processing module for SVG files. It also provides methods for importing and vectorizing raster images. Previous studies such as *Liu et al.* (2020); *Xu et al.* (2007); *Jain et al.* (2015) introduce methods and algorithms that intend to extract, process, and stylize shapes into stencil-ready

shapes using these inputs. These raster image processing procedures expect the source images to provide finalized designs. When users give an image, these systems aim to construct a finalized painting stencil/paper cut by performing a set of actions such as generating bridge shapes to connect island shapes. In comparison, artisans using *InfiniteLayer* could have more diverse goals with images that they are importing. For instance, they could import shapes and use them as bridge shapes instead of island shapes. They might need to process these shapes further or use them as seed shapes for their generative algorithms. Therefore, instead of building an all-in-one vectorizing function that processes images with specific intent (i.e., to create finalized painting stencils), *InfiniteLayer*'s vectorize function focuses on extracting shapes from two-tone images.

B) Methods for Manipulating One Stencil

After creating the first a `Stencil` instance, users can further process this group of shapes by changing their graphical properties. The system provides functions to adjust a path's location (e.g., `MOVE()` / `MOVETO()`) and geometrical outline (e.g., `SCALE()`, `ROTATE()`, `OFFSET()`, `SMOOTH()`, and `SIMPLIFY()`). When users manipulate a stencil using these functions, *InfiniteLayer* will automatically create a new state in the stencil to store the altered copy. Therefore, users have access to a complete history of how a shape is manipulated throughout the design process.

C) Methods for Manipulating Multiple Stencils

Besides modifying the properties of one stencil, users can create intricate designs by manipulating multiple stencils simultaneously. *InfiniteLayer* provides three groups of multi-stencil manipulation methods. To start, users can use `ALIGN()` to

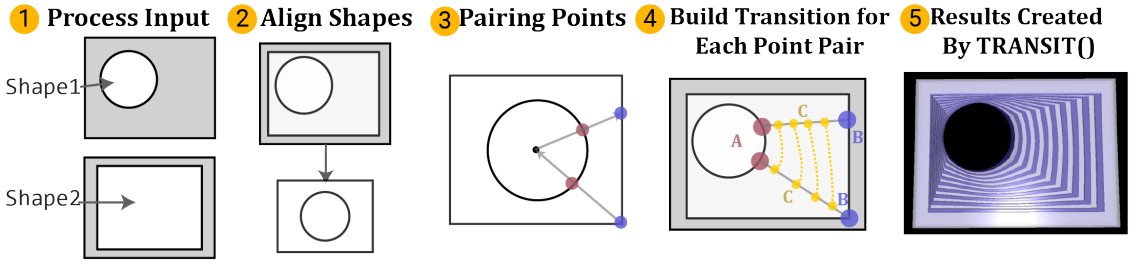

Figure 2.5: The `TRANSIT()` function generates intermediate shapes between two input shapes. *1)* Shape1 is the start shape, and Shape2 is the target shape. *2)* The algorithm temporarily aligns shapes by their centers. *3)* For every point on Shape1, it finds or adds a point on Shape2, so that the line connecting these two points goes through the center. Repeat for points on Shape2. *4)*For every point pair (A, B), the function translates Point A to Point B according to stepCt and the easing function, i.e., finding intermediate points (C). Connecting corresponding C points together to generate an intermediate shape.

adjust multiple stencils' locations. Also, users can perform boolean operations (i.e., polygon clipping operations) on multiple stencils. In addition to these relatively common manipulation methods, *InfiniteLayer* provides `TRANSIT()`, a function designed explicitly for multilayer sculpture. In multilayer artwork such as *Clary* (2013); *Schama* (2019); *Ibbini and Noyer* (2021), I observe that gradually transforming one shape to another over many layers is a commonly used technique. When calling the function `TRANSIT()`, users input three groups of information: 1) two stencils, 2) the number of the in-between steps that they would like to generate (`stepCt`), and optionally, 3) an easing function they would like to use. `TRANSIT()` then produces the in-between shapes. By default, *InfiniteLayer* uses a linear easing function to produce in-between steps. Users can choose other easing functions such as `PolyIn()`, `PolyOut()`, `PolyInOut()`. Alternatively, users can implement easing functions

from scratch. Figure 2.5 illustrates the algorithm I designed for `TRANSIT()` and provides an example generated with the function.

D) Utility Methods

The last category of methods contains utility functions that offer quick access to commonly performed actions. `TrimByBoundary()` and `UnionAllPatterns()` are functions that prepare stencils for the fabrication process. When using machines such as laser cutters and vinyl cutters, it is critical to ensure that all cutting actions are performed on the material because cutting outside of the material can cause serious damage to machines. `TrimByBoundary()` checks all layers and their corresponding stencils to trim off paths that exceed the boundary stencil. If there are overlapping shapes in the design, common fabricators will cut the overlapping areas multiple times, leading to potential damage to the material and machines. `UnionAllPatterns()` ensures that all overlapping shapes are unioned into a connected shape. Furthermore, I provide `RevealAllPatterns()`, a function that ensures every single stencil is visible at the topmost layer. It addresses a design strategy where designers first place key stencils on the individual layer, then gradually ensure that every stencil is visible from the top layer by appending stencils in the previous layer to the next layer. Figure 2.4 provides a sample scenario for using this function.

2.6 Evaluation

I evaluate and reflect the design of *InfiniteLayer* from three perspectives: first, does *InfiniteLayer* fulfill the intended objectives? Second, does the design of the system offer cognitive support for users' creation processes? Finally, what are the

limitations of the tool?

2.6.1 Fulfilling Objectives

I designed *InfiniteLayer* to be a programming-based, vector-graph authoring toolkit that assists the design and fabrication of multilayer sculpture. Specifically, I want to give users control over the design process by *supporting flexible design processes and design goals*, and *assisting prototyping and fabrication.*

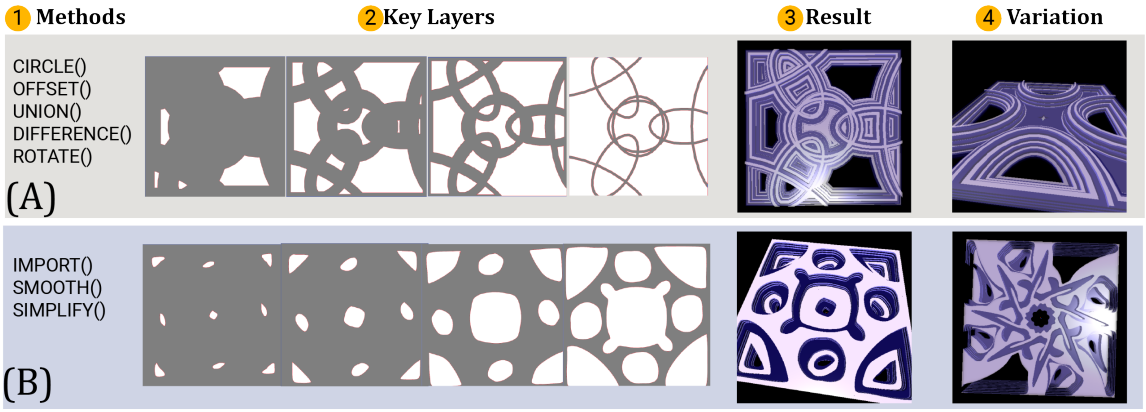

Figure 2.6: *InfiniteLayer* offers flexible support for various design approaches and goals. Design A utilizes a generative algorithm, while Design B relies on vectorized shapes extracted from raster images.

Flexible Design Approaches and Goals:

InfiniteLayer focuses on offering foundational structures and utilities without forcing a specific design approach. In the workflow illustrated in Figure 2.3, most creative tasks happen in step three when users create and manipulate stencil instances. *InfiniteLayer* supports various stencil creation methods, such as SVG importing, raster vectorizing, using shape/line generating functions, and manual authoring.

When manipulating these shapes, users also have different choices. Figure 2.6 displays two groups of designs generated with *InfiniteLayer*. Design A is designed using a generative algorithm. Using the same algorithm, users can quickly create design variations. In comparison, Design B focuses on vectorizing raster images instead of constructing shapes from scratch. While these two designs vary in their approaches and results, *InfiniteLayer* assists both cases by providing a solid structure for starting the designs, interacting with shapes, and exporting these designs.

In addition to utilizing the methods that *InfiniteLayer* provides, users have access to the fundamental data structures that store the path information. Because users can directly manipulate shapes at the point level, they can freely extend *InfiniteLayer* by adding new functions. One advantage that the Python-based toolkit has is the convenient access to the extensive community-contributed library. For users with complex design goals, the ability to import external modules could relieve them from the burden of reinventing wheels. For instance, users who generate their designs procedurally might need access to various random generators that use different underlying distributions. In this case, users can import and incorporate desired libraries into their design pipeline smoothly. Moreover, *InfiniteLayer* does not limit the type of design that users can make. Although *InfiniteLayer* intends to support multilayer sculpture design, users can make single-layer stencil and paper cut for decoration or painting purposes.

Assisting Prototyping and Fabrication:

InfiniteLayer aims to support various fabrication goals. To the best of my knowledge, related design-aid tools currently offer little or no simulation support. Designers

mostly rely on layering 2D transparent shapes to visualize the final output or seeking additional 3D modeling tools to convert their designs to 3D. In these cases, designers face additional 3D design challenges and need to handle repetitive work such as aligning layers and adjusting thicknesses.

InfiniteLayer provides a 3D simulation module that visualizes users' design. By running a convenience method, `exportTo3DSimulation()`, *InfiniteLayer* packs users design into a stand-alone WebGL-based simulator. Users can view and interact with their designs using a web browser without installing any additional software or library. Besides providing a 3D model of users' designs, the simulator offers a built-in control panel that lets users adjusting three major aspects of their design.

To start, users can adjust their models' rendering settings by changing the hue, opacity, rendering mode, and coloring rules. Using these parameters, users visualize how potential material choices (e.g., transparent acrylic board vs. solid paper board) can impact their designs' appearance. The option to render in wireframe instead of solid material can potentially assist users who need to troubleshoot individual shapes. Furthermore, users can adjust the thickness of each layer. The thickness of each layer impacts the design result significantly. With this support, users can visualize, experiment, and control their layered sculptures' depth easily.

Last but not least, users can select the range of layers that they want to visualize. This function helps address the issue that designers often need to imagine and remember how each layer stack on the previous one when assembling the sculpture. For designs that don't have a uniform boundary stencil across all layers, it is essential to have an assembly guide that records each layer's location and orientation. With this

function, users can inspect the assembling process of their design in a step-by-step fashion. Figure 2.7 provides examples of simulations generated using *InfiniteLayer*.

Once users finalize a design, they need to export designs for their chosen fabrication method. Whether they decide to make the design using a laser cutter, or a handheld craft knife, *InfiniteLayer* can prepare files that suit their needs. Exporting designs to SVG would support the majority of cutting-related fabricators, such as laser cutters and die cutters. For users who wish to batch print layers for manual cutting, *InfiniteLayer* provides an all-in-one pdf file. Besides exporting format, users could have specific requirements for how individual layers are rendered. For example, a laser cutter might only recognize paths with a particular color and thickness. Users can adjust the default rendering settings using the commend `stencil.assignAttr()`, which takes a dictionary of style settings such as stroke, fill, stroke-width. Moreover, *InfiniteLayer* provides utility functions so that designers can adjust the basic parameters such as unit and pixel per inch.

In summary, *InfiniteLayer* provides language and tools for designing a multi-layer sculpture without limiting the type of design and the design approaches. Also, *InfiniteLayer* offers a convenient 3D simulation that helps users visualize their creations. By providing multiple exporting formats and access to all rendering settings, I hope that users can create fabrication-ready designs without relying on external simulating and format-converting tools.

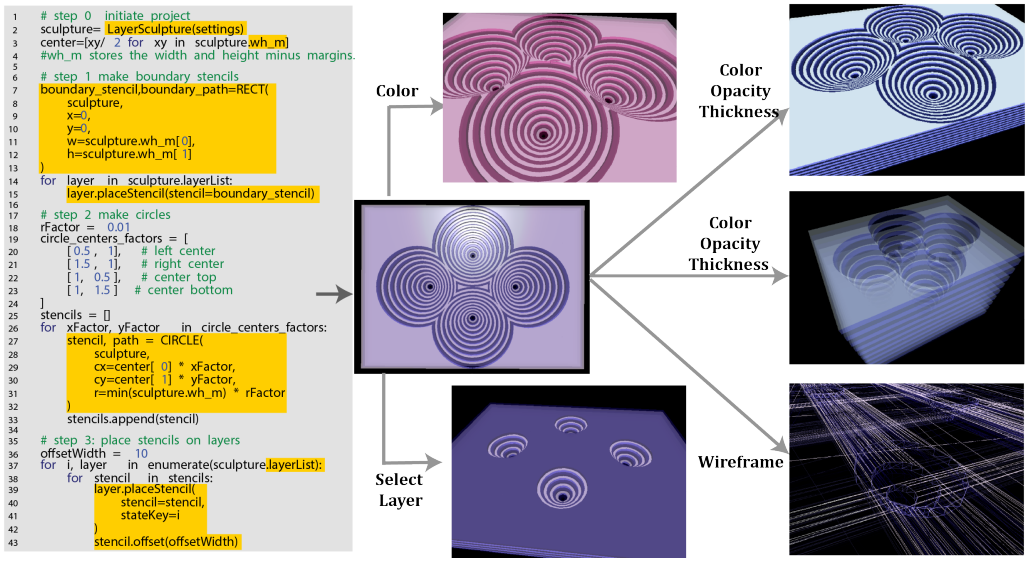

Figure 2.7: A design created with *InfiniteLayer* and its script. I highlight syntax and operations that are specifically supported by *InfiniteLayer*. I also demonstrate various rendering options that users can adjust on the web-based simulation tool that *InfiniteLayer* generates.

2.6.2 Cognitive Support

I use the cognitive dimensions framework introduced by *Green and Petre* (1996) to evaluate how well *InfiniteLayer* assists users at their design task at a cognitive level. Designed to evaluate visual programming environments by capturing cognitively relevant aspects of the programming structure, this framework is a tool to discuss "artifact-user relationships" from thirteen dimensions (*Green and Petre* (1996)). Because *InfiniteLayer* is not a full-fledged programming language but a programming toolkit based on Python, not all dimensions are relevant. I omit these dimensions in my evaluation.

I first reflect how the structure of *InfiniteLayer* helps users tackle a design task by

examining the *Closeness of Mapping*, *Diffuseness*, and *Role-expressiveness* dimensions. The *Closeness of Mapping* dimension reviews how users' tasks are mapped to the programming world. A closely mapped system is likely to lead to a smoother transition from users' domain tasks to operations in the programming environment (*Green and Petre* (1996)). *InfiniteLayer* builds a mapping between the physical structures and digital structures by ensuring that physical structures and their corresponding digital instances have similar names and functions. A `Layer` instance corresponds to a physical layer of material in the physical design. Instead of using "Path" or "PathList," I use `Stencil` to represent the shape to be cut. When using a physical stencil to cut material (e.g., a metal die-cut stencil), designers can move the stencil or alter the stencil to create various cuts. I model and support this process by introducing `State`, a class to hold these manipulated copies of the original shape. I also aim to name functions closely with the physical operations that designers would use if they were to design by hand. For example, to place a stencil on a layer, users would call `layer.placeStencil()`. To align two stencils by their top right corner, users would call `ALIGN(stencil1, stencil2, ''TOP_RIGHT'')`.

The naming of structures and methods also connects to the *Diffuseness/Terseness* dimension, a dimension that examines "the number of symbols required to express a meaning (in the programming environment) (*Green and Petre* (1996))." In general, I try to use compact names for methods and structures, while ensuring that they capture the intended operations (e.g., `.TrimByBoundary()`, `UnionAllPatterns()`, and `ExportToPdf`). In cases where the names tend to be over diffuse,

I provide shortcuts. For example, the cuttable area's width and height are stored in a variable called `width_height_wo_margin`. The system provides a shortcut variable `wh_m` that links to the same value.

Role-expressiveness is another dimension that looks into how a programming environment helps users connect and tackle their tasks. Specifically, *Role-expressiveness* describes how easy it is to read and comprehend a program. Figure 2.7 shows an example and the script used to generate the design. I highlight structures and operations that *InfiniteLayer* provides. Except for shortcut variables (i.e., `wh_m`, which is a shortcut for `width_height_without_margin`), users with Python programming experiences should find the syntax and operations easy to comprehend.

Besides examining structural support that *InfiniteLayer* can offer, I also reflect the potential programming experiences that *InfiniteLayer* support. Specifically, I focus on three cognitive dimensions: *Hidden Dependency, Progressive Evaluation,* and *Visibility*. *Hidden Dependency* describes invisible relationships between components, which are generally undesirable because they might cause unexpected side effects (*Green and Petre* (1996)). In *InfiniteLayer*, I try to expose dependencies by giving users access to inspect and modify all parameters. For instance, to convert a value from one unit to another (e.g., inch to cm), *InfiniteLayer* relies on the variable `pixel_per_inch` (PPI). By making PPI a modifiable parameter, *InfiniteLayer* hint users this dependency, though a detailed explanation for the calculation still needs to be presented through documentation and tutorials.

Visibility / Juxtaposability denotes how easy it is to access a component or to make it visible (*Green and Petre* (1996)). In my system, the most relevant task is

the creation and retrieval of `State` instances. When a stencil is associated with many states, searching through these states might be challenging depending on how users manipulated the stencils and how well users can define the search criteria. For example, moving a stencil to ten different locations generates ten different `State` instances. Retrieving the rightmost state is an easy task, given that I can describe the searching task clearly. Nevertheless, if users transit a triangle to a rectangle over ten steps, searching for the most circle-like state is considerably more challenging. Aiming to increase the visibility of key `States` instances, *InfiniteLayer* has a built-in system for labeling and retrieving states. When a `State` instance is created, *InfiniteLayer* automatically labels it with an index that records the creation order. The labeling system lets users associate these states with custom names (e.g., "`right_most`", "`circle_like`") so that they can retrieve states using these labels (e.g., `stencil.getStateByKey("circle_like")`).

The last dimension that I examine is *Progress Evaluation*, which evaluates whether the programming tool lets users obtain feedback for partially completed programs (*Green and Petre* (1996)). At any stage of the design, users can run their scripts and display their design results by exporting their designs. Alternatively, users can also inspect individual elements within their design. For example, users can print the content of a Stencil instance or check the number of stencils on a layer. Therefore, *InfiniteLayer* does offer partial evaluation support. However, users need to initiate these evaluations. In contrast to a programming environment such as Processing (*Reas and Fry* (2007)) that offers live rendering, *InfiniteLayer* does not provide real-time, automatically updated results and simulations.

2.6.3 System Limitations

As a programming-based toolkit aiming to assist visual designs, *InfiniteLayer* provides convenient and precise control over shapes. For example, because *InfiniteLayer* offers access to all parameters and structures, users can easily adjust shape information at point-coordination levels. While this design gives users a great degree of flexibility, users with a limited understanding of data structures (e.g., array or dictionaries) might accidentally modify these variables (e.g., changing the array's original content while intending to change a cloned copy). Because *InfiniteLayer* is embedded in a full programming language, there are limited safeguards in the current system. This may make the system less approachable for users with limited programming experience. Future versions of *InfiniteLayer* may utilize a true domain-specific language (DSL) or a graphical interface.

While *InfiniteLayer* supports a wide range of designs, I also acknowledge that the programming-based vector design pipeline might be more suitable for certain design tasks. *InfiniteLayer* excels at supporting designs that require repetition, precise manipulation, and generative algorithms. Users would likely find it relatively easy to create artwork similar to the styles of *Vantours* (2019); *Clary* (2013); *Ibbini and Noyer* (2021). In comparison, authoring shapes that resemble objects realistically (e.g., *Tomsky* (2020); *Kubo* (2009); *Vantours* (2020)) from scratch using programming can be considerably more challenging than using tools with graphical interfaces. In these cases, designers can still use *InfiniteLayer* as a simulation tool that imports and processes drawings, but creating realistic objects without image input would be difficult.

The simulation method I provide also exhibits several limitations. I designed and implemented the simulation using web-based technology because it is a cross-platform and out-of-box solution. It also enables various simulation features to quickly alter the 3D model (Figure 2.7). Nevertheless, rendering a large number of intricate shapes using WebGL could take a significant time. To reduce web rendering time, users can choose to render layers in batches. For example, users can render layers 0-20, 20-40, 40-60 instead of rendering all sixty layers at once. In the future, I hope to add another simulation module that generates 3D models in standard formats such as STL and OBJ. Although users would need additional software to view and modify these models, having a fully rendered model without significant rendering time could be desirable. Additionally, *InfiniteLayer* currently simulates designs using fixed settings for lighting (direct light from above) and material (a plastic-like shining material). In the future, I hope to offer more options so that users can render in the material and lighting that resemble their real-world design.

2.7 Discussion

Because the *InfiniteLayer* is a programming-based tool that assists the design of physical artifacts, it faces unique challenges associated with building creativity-support tools, supporting programming-based control, and connecting digital design and physical fabrication processes. In this section, I reflect on approaches that I took to tackle some of these challenges and provide insights for building similar systems.

2.7.1 Extracting and Refining Domain-specific Knowledge

To support the design of a specific type of art & craft, tool designers need to understand the components, opportunities, and challenges within the target craft. Depending on the type of craft, tool designers can obtain this domain-specific knowledge from previous literature, tools, current practitioners, and personal experience. Collecting this information is challenging, but processing domain-specific information into meaningful, organizational, and programmable structures is even more difficult.

A craft domain can contain complex components and constraints that require a large number of notations to represent. For example, existing studies describe many multilayer art-related techniques with different abstraction levels: "wrap", "wrinkle", "expand", "erasing", "hollow out" are all terms and operations related to paper cutting art (*Ryan and Avella* (2011); *Igarashi and Igarashi* (2010); *Liu et al.* (2020)). Directly adopting these terms without identifying core actions and meanings behind these terms is problematic.

I approach this challenge by identifying the "grammar" of the craft, a set of core components and constraints that designers follow when "writing" a design in this craft domain. The notion of grammar comes from studies in culture conservation (e.g., the Bailey-Derek Grammar: (*Noel* (2015)) and information visualization (i.e., the Grammar of Graphics (*Wilkinson* (2012)). Intending to identify a set of grammar for a craft, system designers need to focus on synthesizing the minimal set of components and rules (e.g., how nouns and verbs work together) before expanding to more detailed structures (e.g., how to choose among "a", "an" and "the").

In this work, I demonstrate how the set of grammar (e.g., layers and stencils)

becomes my tool's core structure and interface. Additionally, this grammar-driven structure opens opportunities for future extension and refinement of the tool.

2.7.2 Supporting the Design and Fabrication of Physical Art & Craft with a Programming-Based Toolkit.

Designing physical artifacts through programming has become an increasingly common design approach as designers, researchers, and tool builders gain more understanding of the opportunities and constraints associated with this approach. As described by *Jacobs* (2013), computation design is a way to apply "procedural thinking to a design task", which brings many potential benefits such as supporting high levels of precision and complexity, also enabling algorithm-driven, parametric designs. Besides generic programming-based tools like *Processing* and *Openframeworks*, craft-specific such as *PEmbroidery* (*Levin et al.* (2020)), *Dynamic Brushes* (*Jacobs et al.* (2018)) are also revolutionizing how artists can approach a design task.

While offering unprecedented opportunities, programming-based design-aid tools also have unique challenges related to their target user, interface design, and connection to physical fabrication processes. A common concern is that programming-based systems require their users to receive technical training (*Jacobs* (2013)). Tools like *Dynamic Brushes* utilize block-based programming to address this issue (*Jacobs et al.* (2017)). Whether to provide a visual interface and how to design these interfaces remain to be open questions (*Jacobs* (2017); *Jacobs et al.* (2017, 2018); *Li et al.* (2020)).

In addition to design suggestions and guidelines provided by previous studies

such as *Jacobs* (2013, 2017); *Jacobs et al.* (2017, 2018); *Li et al.* (2020), I want to highlight the importance of offering fabrication support. Because users' ultimate design goals likely include creating physical artifacts, tool designers need to consider how digital designs are transformed into physical objects. System designers should review common fabrication methods within the domain to develop solutions targeting these approaches. For example, systems might provide support for SVG exporting if the craft is related to laser cutters. Nevertheless, design-aid tools can still struggle to meet the constantly-changing fabrication needs because fabrication technologies rapidly change. For instance, specific machines might only recognize files rendered a particular way; specific file formats can go out of trend. As a result, designers might need to develop expertise in a wide range of tools to complete one design task: e.g., creating shapes in tool A, adjusting color settings using tool B, and converting file format using tool C.

Instead of aiming to solve all fabrication needs at once, I tackle this challenge by making the *InfiniteLayer* an easily extendable tool. To ensure that users can easily extend the tool, I offer a compact system infrastructure and support a core set of techniques, and at the same time, providing users easy access to all parameters and data structures. My choice of language (Python) also helps users to import and implement modules quickly.

2.7.3 Future Directions

A dedicated toolkit that supports the design and fabrication of multilayer sculptures through programming can facilitate innovative techniques and applications of

this art form. Currently, I primarily tested my design using a laser cutter and construction paper. In the future, I plan to test my design outputs using a variety of fabrication methods and material to examine additional design tasks that new combinations of fabrication approaches and material might bring. I also plan to expand this study by collecting additional user feedback and designs. Moreover, I hope this work facilitates future discussion in building programming-based art & craft design tool, and contributes to the conversation about how the programming-based artifact design approach connects and contributes to the research-through-design community from a HCI perspective (e.g., *Mikkonen and Fyhn* (2020); *Gaver* (2012)).

2.8 Conclusion

In this chapter, I provide analyses and a solution to common challenges that multilayer sculpture designers face. By examining existing studies and synthesizing design techniques, I discuss the need for a toolkit that offers prototyping and fabrication assistance while supporting computational design approaches. I contribute *InfiniteLayer*, an open-source toolkit that contains core structures and methods that help users draft, manipulate, and visualize their designs. *InfiniteLayer* displays all three primary characteristics of GCDTs as described in section 1.3.1. The development process of *InfiniteLayer* is also a representative case that starts by gathering domain-specific information. By showcasing the wide range of artifacts created using *InfiniteLayer*, I demonstrate the potential of programming-based design-aid tools and offer insights for designing such systems.

CHAPTER III

Creative Mark-Making Tool Design

Predicting artists' needs is challenging because artists' goals can be diverse, personalized, and unquantifiable. For developers of design-aid systems, there are additional layers of system design and engineering challenges associated with the technologies they have at hand. For instance, with the rapid development of fabrication technology, the fabrication of a particular craft might require a new set of constraints or new ways of assistance. As artists explore the design space of their craft domains, they might also identify innovative design directions that the current tools cannot support. Therefore, it is difficult to treat the development of a design-aid tool as a one-time task.

GCDTs' grammar-driven structures assist a gradual development pattern. To produce a functional tool, system developers need to obtain an essential set of grammar. If tool developers and users identify additional tasks that they would like to support, they can add new elements to the grammar, as long as the additions do not conflict with the existing ones. In this chapter I present the development process of

a GCDT in the domain of mark-making tool design. In this project, I first identify core grammar by analyzing a large set of mark-making tools. Then I gradually add three different fabrication modes to support various design tasks.

3.1 Introduction

The lack of flexibility is one of the major challenges that are associated with domain-specific tools. Because they are designed for a specific set of tasks, users with alternative design goals would have to find workarounds. To offer support for a variety of design tasks, GCDTs are extendable. The extendability associated with GCDTs is twofold. To start, users can extend the grammar as long as the additions are compatible with the existing grammar. Then, GCDTs let users add additional modules. This chapter demonstrates a GCDT named *MarkMakerSquare* that supports the design and fabrication of creative mark-making tools.

Mark-making tools are essential to visual artists regardless of their choices of the medium. For example, painters create images with various pens, brushes, and markers. To write in different styles, calligraphers swap their copperplate nibs. When making 3D forms, sculptors and ceramic artists shape their material and make impressions using stamps, scrappers, and shapers. The creation of marks is a prevalent activity in everyday life. Writing instruments such as pen and pencil are among the most common tools people access daily. As a result, there are countless variations of mark-making tools available.

Many factors, such as material, size, and even handle design, influence a mark-making tool's functionalities. Depending on the purpose of mark-making activities

and the target users, these instruments' design prioritizes different factors. For instance, markers and crayons designed for children focus on creating easy-to-grab yet hard-to-swallow shapes while having relatively flexible requirements for making precise marks. In comparison, high-end watercolor brushes made for professional artists focus on using fibers that hold enough water and remain in shape so that artists can create consistent marks. While brush for painting job and brush for glazing ceramics might share similar bristle design, they make entirely different marks because of their material.

Despite the extensive variations of mark-making tools available, artists continue to develop new tools. The need for personalized mark-making tools is common for artists across many fields because artistic creation goals are highly diverse and personalized. For example, watercolorists use toothbrushes and sponges to create splashes and blobs (e.g., *MacKenzie* (2010)). Printmakers repurpose credit cards to make customized scrappers (e.g., *Ayres* (2001)). Painters cut their brushes into specific shapes to paint parallel lines.

Besides modifying existing tools and repurposing other objects, artists can also make their tools from scratch. Although making tools from scratch enables a broader range of design possibilities than the modify/repurpose strategy, designing and making personalized mark-making tools can be difficult for several reasons. To start, artists need to have relevant design skills such as drafting and modeling. Artists also need training in physically making the instrument. For instance, to manually create a brush, artists need to know about woodworking (to make the brush handle) and fiber handling (to glue and shape the bristle). Most importantly, navigating through

a large number of potential design choices is challenging (e.g., synthetic fiber vs. natural fiber, round tip vs. flat tip).

Computer-aided fabrication methods offer opportunities for the designing and making of personalized mark-making tools. With the rise of maker culture and advances in fabrication research, CNC machines such as 3D printers and laser cutters become increasingly available to individuals (*Tanenbaum et al.* (2013); *Willis* (2018)). Besides offering relatively low-cost and efficient ways to fabricate customized artifacts, the computer-aided fabrication method also supports detailed and complex designs. For example, artisans can build 3D models for pen nibs, test prototypes using 3D printers, and finally, cast with metal. Despite these advantages, designing mark-making tools remains difficult because designers still face a vast and undefined design space, where they need to leverage a considerable number of factors.

To address these challenges in designing and fabricating customized creative mark-making tools, I examine an extensive collection of mark-making instruments. The collection includes various pens, markers, crayons, brushes, scrapers, nibs, and paint sticks. After summarizing key factors that impact these instruments' functionalities, I test various fabrication methods such as 3D printing, casting, and manual construction. Finally, intending to let users explore different design attributes and fabrication methods without extensive training, I build a programming-based toolkit to supports the design of creative mark-making tools and demonstrate a wide range of marks that they enable.

In this chapter, I contribute analysis and solutions to the challenge of designing and fabricating personalized creative mark-making tools. I contribute an open-source

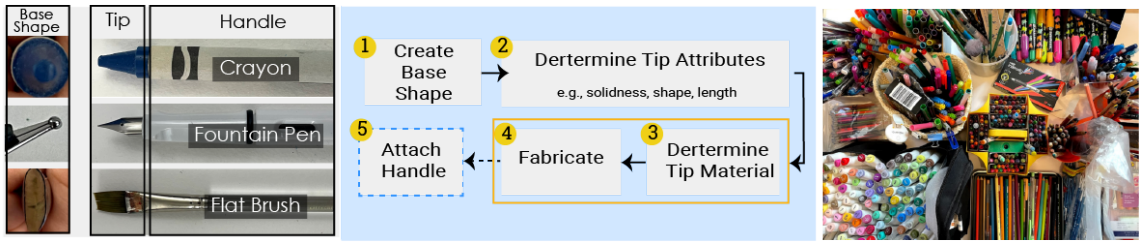

Figure 3.1: (left): The anatomy of mark-making tools.
(Middle): A five-step pipeline for designing/making mark-making instruments. Step 3 and step 4 are closely related because different tip material choices lead to various fabrication methods. Step 5 is optional for some mark-making tools if tips and handles are fabricated with the same material and process (e.g., a crayon).
(Right): A subset of mark-making tools that I collected.

toolkit, *MarkMakerSquare* to support the design and simulation of a wide range of mark-making instruments. Although the toolkit can support various types of instrument design, it excels at designing precise, complex, and creative mark-making instruments. Designers who are interested in experimenting with unconventional mark-making instruments would find this toolkit particularly useful. I present a collection of creative mark-making tools designed using my toolkit and examine various marks that these instruments can support. Finally, I reflect on several challenges I encountered while developing tool-designing systems and share my strategies for tackling these challenges.

3.2 Related Work

I review existing research related to creative mark-making tools and associated fabrication methods in this section to synthesize basic knowledge for designing and fabricating mark-making instruments.

3.2.1 Mark-Making Tools

Mark-making is "the creation of a perceived anomaly, or felt difference, on or in a surface.(*Malafouris* (2021))" While I consider any object that makes marks a mark-making tool, I focus on designing mark-making tools for creative activities, in contrast to creating tools for everyday writing or decoration purposes (e.g., *Grishkoff* (2020)). Also, I consider the design of tips as my main task because they determine the marks that a tool can make. Therefore, I intentionally omit handle design discussion from this examination.

I primarily examine tools used in painting/drawing, printmaking, and 2D/3D surface decorating. In each of these usage scenarios, there are wide ranges of needs for mark-making tools. For example, in watercolor painting, the choice of brushes is critical. Watercolorists often need a collection of brushes with different fibers, tip shapes, and sizes (*MacKenzie* (2010)). In printmaking, artists typically need a wider range of tools that are made with different material. In addition to various brushes, printmakers would need metal tools such as metal roulettes (i.e., rollable wheels with patterns on them), silicone scrapers, and palate knives (*Ayres* (2001); *Leaf* (1984)). Mark-making tools vary significantly across different genres of calligraphy. For instance, Chinese calligraphy features long-tip brushes traditionally made with animal hair (*Chiang* (1973)), whereas copperplate calligraphy is associated with metal nibs (*Winters* (2014)).

Despite a wide range of existing tools in these domains, artists still need to develop and customize their tools. Sponges, sticks, and toothbrushes are ordinary objects that artists can repurpose as mark-making tools in painting and printmaking

(*MacKenzie* (2010); *Leaf* (1984)). To make creative marks, printmakers also modify everyday objects such as plastic doilies, cloth, credit cards, and natural material such as dried flowers and leaves (*Ayres* (2001)). In addition to these everyday objects, artists find creative solutions to support specific techniques. For example, when creating mezzotint[1] plates, burnishers available on the market do not always satisfy artists' needs. As a result, artists use dental tools as precise burnishers (*Wax* (1996)).

Different tools are associated with their unique components and terminology (e.g., tines in nibs, bristles in brushes), I broadly consider mark-making tools to have two major components: handle and tip (see Figure 3.1). Handles and tips can be made with the same material (e.g., crayon, oil/pastel stick) or have detachable designs (e.g., dip pen). Artists will primarily use the tip section to make marks. I use the term "base shape" or "tip base shape" to describe the shape that connects the handle section and the tip section. While many considerations go into the design of a handle, this chapter primarily focuses on the design of tips because they determine the final marks that a tool can make.

3.2.2 Making Mark-making Tools

The making process of mark-making instruments is often documented with the purpose of culture/history preservation. For example, Thangka painting is a traditional Tibetian painting style that uses unique brushes. The process of making bamboo brushes is documented so that Thangka artists can recreate these bamboo

[1]Mezzotint is a intaglio printmaking technique. Artists use a heavy metal tool (known as "rocker')' with many sharp points known as "teeth." They make repetitive marks by pressing the rocker on the plate.

brushes following the traditional method (*Jackson et al.* (2006)). Work that documents the history of iconic tool manufacturers can also include the design and process of various tools (e.g., *McKinney* (2018)).

There are a few early literature that document the process for contemporary brush making. They tend to focus on developing terminologies for brushes (e.g., *Dickinson* (1943)) or discussing fibers suitable for brushes (e.g., *Kirby* (1950)). Examples such as *Neddo* (2015) serve as "cookbooks" for artists to start experiments, as they offer tips on making tools such as charcoal sticks, simple dip pens, and twig brushes.

Because I aim to support a wide range of mark-making tools, I cannot directly adopt creation pipelines for a specific type of mark-making tool. For example, the process for designing a charcoal stick (as described in *Neddo* (2015)) is significantly different from the fabrication pipeline for a Thangka bamboo brush (*Jackson et al.* (2006)). As a result, I synthesize a high-level designing/making pipeline for mark-making tools to be a five-step process (Figure 3.1). To start, artisans design the base shape of the tool. Then, they decide on various attributes of the tip section. For example, they select the shape and length of the tip. With a tip design, artists can then decide the material of the tip. The selection of the material determines the fabrication process. For example, to make a crayon, one needs to cast pigmented wax using a mold. In comparison, brush making requires more manual work, as artisans need to bundle fibers together. After creating the tip, artisans join the tip section and the handle section together. The handle attachment might be optional for some tools. For example, crayons' handles and tips are often cast as one piece.

In addition to work focus on examining fabrication strategies/processes of mark-

making tools, I also look into computerized fabricators in the domain of art & craft support. Overall, there are many examples where fabricators such as 3D printers and laser cutters play central roles in supporting art & craft making (e.g., *Torres et al.* (2016); *Jacobs* (2013); *Mueller et al.* (2013); *Hudson* (2014); *Iarussi et al.* (2015b)). Fabrication-related literature and design-aid systems provide insights into the design of my tool.

3.2.3 Digitizing Mark-Making Tools

Two categories of literature enhance my understanding of mark-making tools. The first category of research focuses on examining the properties of physical brushes and digital building models. For instance, by developing physical brushes and analyzing how brush tips move across surfaces, previous work such as *Chu and Chiew-Lan Tai* (2002); *Baxter and Govindaraju* (2010); *DiVerdi* (2013); *Jeng-sheng Yeh et al.* (2002) develop algorithms for simulating brush strokes. The second category of research discusses the use of brushes in robots. Aiming to create physical art pieces, such as calligraphy work (e.g., *Mueller et al.* (2013); *Fenghui Yao and Guifeng Shao* (2005)) and watercolor (e.g., *Scalera et al.* (2019)) using robots, these work document maker-making tools and methods for precisely control these tools. Nevertheless, almost all work in both categories focuses on brushes and has a limited discussion for any other type of mark-making tool.

Figure 3.2: Six major factors impact the functionalities of a mark-making instrument.

3.3 Design Space

To assist the design of mark-making tools, I first need to understand critical factors that designers need to consider when approaching a design task. In other words, I need to identify attributes that designers can modify in a tool-making process.

Existing research discussed in Section 3.2 builds a foundation for identifying key factors that impact brush designs. However, because of the lack of existing research that reviews tools other than brushes in detail, I cannot directly adopt these design factors. As a result, I gathered and examined an extensive collection of mark-making tools that include various pens, markers, crayons, brushes, scrapers, nibs, and paint sticks.

In this analysis, I focus on examining attributes and properties that directly impact the marks that a tool can make. There are important attributes that can affect mark-making results indirectly. For instance, the choice of mark-making medium

certainly affects design choices significantly. A watercolor brush and an oil painting brush can differ considerably because of their designed mark-making medium. Nevertheless, the choice of the medium is not a property of the tool but an indirect design consideration that impacts many physical attributes of a tool.

Figure 3.1 displays a subset of tools that I collected. By analyzing this collection of mark-making tools, I identified six factors that impact mark-making tools' capability and performance. This section describes these six factors and discusses how the combinations of these factors enable a wide range of mark-making tools. Figure 3.2 provides an overview of these six factors.

3.3.1 Solidness of the Tip Section

Solidness describes the number of parts that the tip section contains. Because artists use the tip section to create marks, the number of parts that touch the surface directly impacts a tool's functionality. Tools like ball pens, fine liners, and crayons typically have only one surface-contacting part, whereas pen nibs might have two tines that can split when adding pressure. Brushes typically have many loose tips, and each strand of fiber can leave a mark by itself.

3.3.2 Tip Angle

Tip angle is the designed holding angle of a tool. In many types of mark-making instruments, users can create marks from any angle. For example, ball pens and pencils can leave marks at almost any angle, as long as they touch the surface. Brushes can also leave marks with adjustable angles, though different holding angles

might lead to various marks. Some tools have designed ranges of holding angles. To create the broad and flat marks that highlighters and chisel-shaped markers are designed for, users need to hold them at a specific angle. These tools are often equipped with several "ideal" angles to make several different types of marks by changing the holding angles. Last but not least, some tools only function at their designed holding angles. Fountain pens, dip pen nibs, parallel pens, and stamps are examples where users need to adjust their holding position to make marks.

3.3.3 Shape

The shape of the surface-contacting part determines marks that a tool can make. To provide a more in-depth language for describing shapes within the tip section, I consider the base shape and tip shape as two different parameters that tool designers can adjust. The base shape describes the shape that connects the tip section to the handle section. For example, many pens, pencils, markers, and drawing sticks use a circle as their base shapes. Ovals and rectangles are also common shapes.

In addition to base shapes, tool designers can also alter the tip's shape, which describes how various surface-contacting parts are grouped together. For instance, filbert, round, and flat are several common tip shapes available for watercolor brushes. It's worth noting that the base shape and tip shape can be completely independent. For example, a round brush and a precise crayon can share the same base shape (i.e., circle) while having utterly different tip shapes.

3.3.4 Base Size

Base size describes the size of the base shape. In many existing tools, manufacturers offer a set of similar tools that only differ from each other by base size. For instance, artists typically have various sizes to choose from when selecting calligraphy brushes and permanent markers.

3.3.5 Length

The length of the tip also impacts the marks that a tool can make. The impact is especially salient when the tip material is soft. For example, a long crayon might function the same as a short crayon, whereas a long brush creates completely different marks compared to a short brush.

3.3.6 Material

Lastly, the choice of material for the tip section is a crucial design factor that tool designers need to consider. Depending on the purpose of the mark-making tool, designers would choose different material. Common material such as metal, fiber, silicone, wax, clay, sponge, and plastic are used in many tools. The selection of material determines the fabrication method of the tool. For example, designs that use castable material such as metal, silicone, and wax require designers to create cast molds of the tip design. When developing tools with fibers, the primary design task is to design structures that can hold fibers.

In summary, six factors impact the functionalities of a mark-making instrument. However, experimenting with these factors could be challenging as tool designers

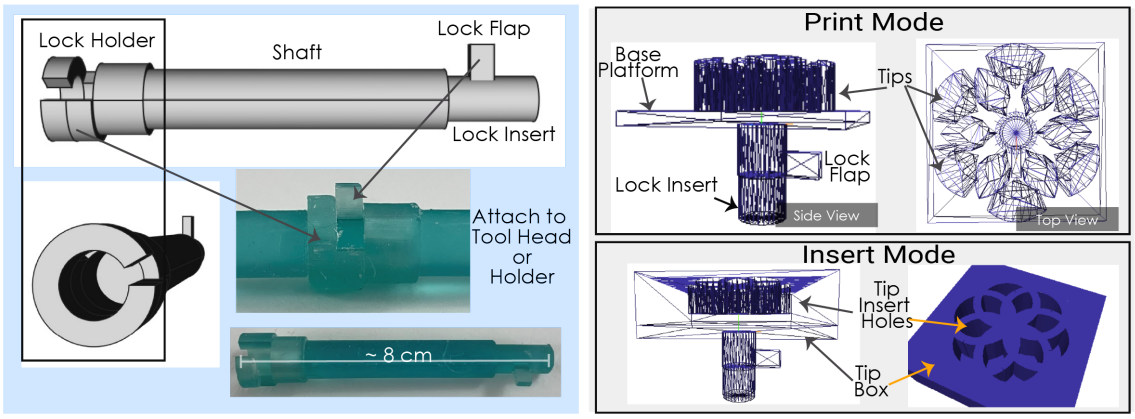

Figure 3.3: (Left): The design of a generic handle that features interlocking components.
(Right): *MarkMakerSquare* supports three fabrication modes. The print mode generates designs that can be directly 3D printed (top). The insert mode generates designs that need to be printed and manually assembled.

need to modify, remake, and test their models repeatedly. Ideally, I hope there is a design-aid tool that can let users quickly experiment with different factors.

3.4 Designing and Fabricating Tools

To support the design of mark-making tools, I developed a python-based toolkit called *MarkMakerSquare*. I designed *MarkMakerSquare* with several objectives. The primary goal is to provide a set of viable solutions for making customized mark-making tools. More specifically, I focus on supporting the design of tip sections rather than the design of handles. *MarkMakerSquare* lets users build and customize the tip section's designs and provides a simple solution to a universal handle design. Figure 3.3 illustrates the generic holder that *MarkMakerSquare* offers. The holder

features interlocking components so that users can attach different tool heads. The interlocking structure also makes the holder stackable, as users can print multiple handles and join them quickly. Also, users can adjust the length of the holder by modifying the provided 3D model directly.

In addition to focusing on tip designs, I designed *MarkMakerSquare* to be a toolkit that lets users explore different tip attributes and fabrication methods. Ideally, users can quickly play with the six design factors summarized in Figure 3.2. Also, when users have many material choices, one fabrication method cannot satisfy all fabrication needs. Therefore, *MarkMakerSquare* offers support for three fabrication modes and is extendable. This section describes *MarkMakerSquare* by presenting its fabrication mode, the creative pipeline that it supports, and the interface that it offers.

3.4.1 Three Fabrication Modes

MarkMakerSquare offers support for three different fabrication modes. While all fabrication modes lead to a 3D model to be fabricated with 3D printers, each mode intends to generate designs for a subset of mark-making tools.

Print Mode:

The print mode supports the design of mark-making instruments similar to nibs, stamps, and scrapers. This set of tools features tip designs that are the extrusions of a base shape. Users can directly 3D print this type of instrument. Figure 3.3 illustrates various components associated with tools in this category. For example, to build a stamp, the users will supply a base shape extruded to the target height

and serves as the tool's tip. *MarkMakerSquare* generates supportive structures such as the base platform and the interlocking mechanisms automatically according to users' settings.

Insert Mode:

When designing brush-like tools, making a structure to hold fibers is the primary design task. The insert mode supports the design of these "fiber holders." There are various ways to secure fibers. Fastening fibers with a ferrule is one strategy. Attaching fibers to a structure with a designated "root" space is another strategy. *MarkMakerSquare* takes the latter strategy and offers solutions that feature a box structure that is carved with holes for fiber insertion.

The process for making such a design includes three key steps. First, users input the base shape. Then, *MarkMakerSquare* produces a 3D model that users can adjust and print with a 3D printer. Lastly, users assemble the mark-making tool by inserting their choices of fibers into the printed structure. Figure 3.3 displays an example generated in this mode.

Cast Mode:

The cast mode supports tools that use castable material. If users intend to build silicone, metal, or wax tools, the main design task is to create a mold to pour material and cast the material into the designed form. For different casting material and design purposes, there are various strategies for building mold (*Cannon* (1986); *McCreight* (1994)). *MarkMakerSquare* implements a two-piece design that has a base structure and a box-like mold structure (see Figure 3.4). The base structure contains hollow spaces that serve as material pouring holes. The tip mold has the

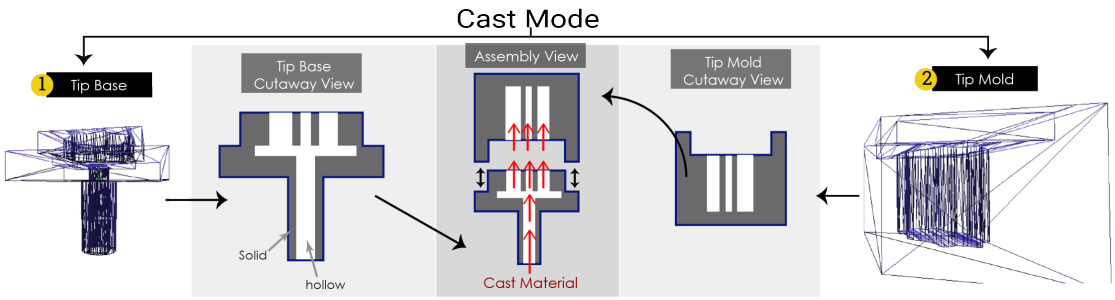

Figure 3.4: The cast mode generates two major structures: the tip base (leftmost) and the tip mold (rightmost). Users can 3D print these structures and join them together for the casting process.

target form to cast. After printing both structures using 3D printers, users will join the mold part with the base structure and cast the tool with their choice of material.

3.4.2 The Creation Pipeline

MarkMakerSquare is a Python-based toolkit that supports a creation pipeline aligned with the generic creation pipeline discussed in Section 3.2 (Figure 3.1). To start a design, users will input a base shape for the tip of the tool. Users currently have two options for inputting the base shape. First, they can supply a list of path information (i.e., point coordination). *MarkMakerSquare* offers a set of utility functions to assist the modification of paths in this format. For example, users can easily convert units, transform (i.e., rotate, scale, translate, and skew) the path, simplify the path. *MarkMakerSquare* also offers a group of shape-generating methods to assist the making of basic shapes and lines such as polygons and bezier curves. Users can take advantage of a programming-based design approach and create detailed and complex base shape designs.

Users might prefer manually drawing the base shapes instead of specifying shape information through scripts. *MarkMakerSquare* offers a set of utility functions that let users convert and process raster images into the ideal path format to support this shape-generating preference. Using these utility functions, users can create their base shape with any raster image authoring tools such as *Adobe Photoshop* and *Procreate*, then import/modify these designs into *MarkMakerSquare*.

After supplying the base shape, users can input settings that modify the model's design details. Users can customize all parameters used to generate these 3D models by adjusting these settings. For example, users can adjust the lock insert's radius, the minimal printing thickness that their 3D printers support, and the tip insert holes' depth. Users can refer to the default values that *MarkMakerSquare* is providing.

With the base shape and setting inputs, users can trigger the exporting commands offered by *MarkMakerSquare*. The primary output of *MarkMakerSquare* is a standalone website that contains the design. The website provides a WebGL-enabled simulation of the target design and interactive widgets that let users change some design factors. Users can experiment with various settings such as trip length, solidness, and rendering method. Then, they can export the 3D model and move it to the fabrication process. In addition to generating the web interface, *MarkMakerSquare* can also export a set of processed base patterns in SVG format. I offer these processed patterns that can provide insights for troubleshooting the design.

The last step in the creation pipeline is physically fabricating the design. Depending on the fabrication mode that users choose, they will use digital fabrication and manual fabrication. In the print mode, the user can directly print the model.

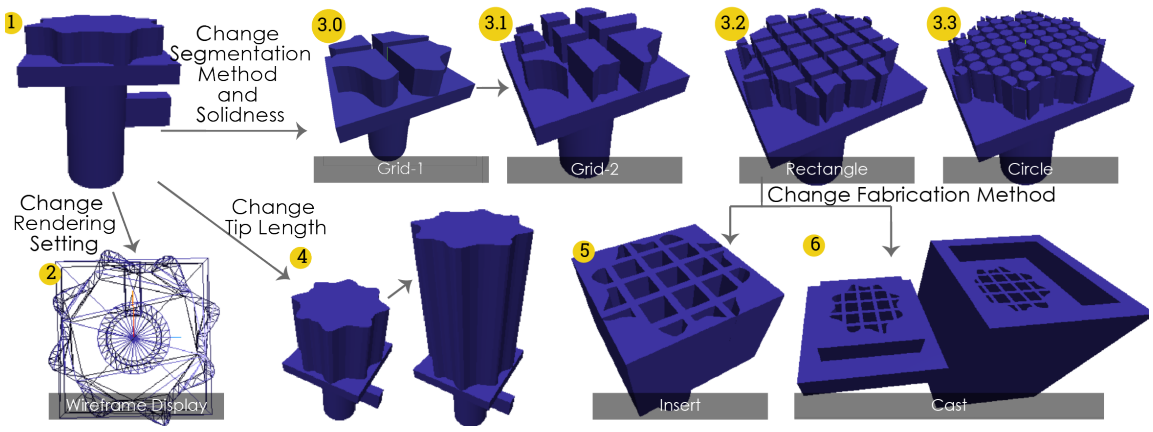

Figure 3.5: The WebGL-enabled interface generated by *MarkMakerSquare* lets users experiment with different design factors. Through the interface, users can 1) modify rending settings, 2) change segmentation method and solidness, 3) adjust tip length, and 4) select the fabrication method.

In the insert and cast mode, the user will print the tip base or the tip mold and manually assemble/cast the design.

3.4.3 The MarkMakerSquare Interface

MarkMakerSquare generates a standalone web interface that lets users navigate various design factors discussed in Section 3.3. At a high level, the interface supports the model's modification in four ways: 1) rendering setting, 2) pattern segmentation method and solidness, 3) tip length, and 4) fabrication method. Figure 3.5 provides examples of these modifications. To start, users can toggle between wireframe rendering and solid material rendering. Here I provide detailed documentation of the other three modifiable areas.

Pattern Segmentation Method and Solidness Adjustment:

As discussed in Section 3.3, two mark-making tools that share the same base pattern could have completely different capabilities if they have different solidness (number of parts touching the surface). *MarkMakerSquare* automatically segments the base pattern into subpatterns. A higher degree of solidness leads to less segmentation (fewer parts), and a lower degree of solidness leads to more segmentation.

MarkMakerSquare currently offers three segmenting methods: segment by grid, rectangle, and circle. The grid segmentation method essentially cut the base pattern using horizontal and vertical lines. When the solidness setting is high, there are fewer cutting lines. When the solidness setting is low, there may be too many cutting lines so that the majority of the original pattern is lost. In this scenario, users can adjust the thickness of these cutting lines.

The rectangle and circle segmenting methods are similar. Both methods essentially fill the entire pattern with the target shape (i.e., rectangle/circle) with specific sizes. A higher degree of solidness leads to larger target shapes, whereas a lower degree of solidness utilizes smaller target shapes. These two segmenting methods are handy when designing tools for the insert mode because segmented parts can serve as the tip holding structures.

Tip Length Adjustment:

The interface provided by *MarkMakerSquare* offers a slider for adjusting the length of the tip. By dragging the slider, users can quickly visualize how the length of the tip impacts their designs. By default, the interface offers a ten-step adjustment for this parameter with an increment of 2mm in each step. Users can adjust the step increment in the setting input stage. In the insert mode, the slide becomes

inactive because the tip's length is controlled by users manually when they prepare their choice of fibers.

Fabrication Mode

Finally, the web-based interface generated by *MarkMakerSquare* lets users quickly switch between different fabrication modes through a drop-down menu. The default mode is the print mode. When switching to the cast mode, *MarkMakerSquare* generates the top mold and the tip base structure side by side.

In summary, *MarkMakerSquare* is a dedicated toolkit that aids the design of creative mark-making instruments. *MarkMakerSquare* implements a design pipeline aligned with the general mark-making tool creation pipeline to support multiple fabrication methods and various material. *MarkMakerSquare* offers multiple utility functions and a web-based interface that lets users create innovative designs without extensive training.

3.5 Showcase

I demonstrate the capability of *MarkMakerSquare* by presenting five groups of designs generated using this toolkit. Each group focuses on delivering or comparing a specific set of features that *MarkMakerSquare* supports. I 3D printed these designs using a hobbyist-level resin 3D printer. I used the water-washable photopolymer resin manufactured by *ELEGOO*, which creates rigid structures that are difficult to break manually if the form is thicker than 5mm.

After physically fabricating these creative mark-making tools, I also demonstrate painting samples made with these tools. I created these painting samples using

calligraphy ink. The procedure for generating these painting samples is as follow:

1. Load the ink once and paint a straight line with the designed holding angle until the ink runs out.

2. Load the ink and paint a straight line with an alternative holding angle (e.g., paint with the side of the tip) until the ink runs out.

3. Load the ink, then paint a scribbling line to demonstrate the marks created in a curve-drawing scenario.

4. If the tool is a printed tool and has an interesting pattern design, load the ink and press the tip's top surface to test the stamp-like functionalities this tool has.

5. Making marks in freestyle

3.5.1 Case 1: Fibonacci Pattern

In this design case, I test a group of tools made with the print mode. I examine the differences created by adjusting that segmenting methods, solidness, and tip length. The base pattern I used consists of a series of rectangles with the width following the Fibonacci sequence. In addition to printing the tool with the highest degree of solidness (without any segmentation), I also printed tools with various degrees of solidness and segmenting methods.

Figure 3.6 displays the printed tool and the marks that they make. The tool with the highest solidness generates large blobs of shapes at the beginning of the stroke

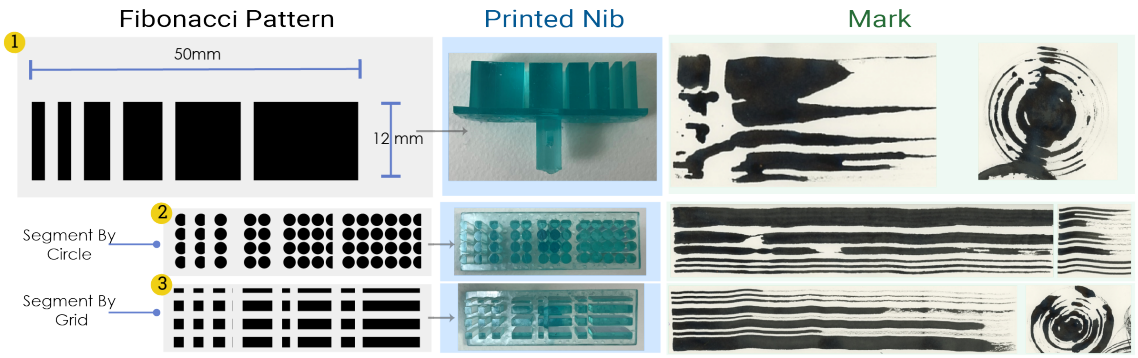

Figure 3.6: Design case 1: tools fabricated with print mode using the Fibonacci pattern.

but quickly runs out of ink. In comparison, tools with lower solidness release the ink more stably. This difference demonstrates that increased surface area contributes to the ink-holding abilities in these tools. I also printed this design with multiple tip lengths (10mm, 13mm, 23mm) but did not observe significant changes in the marks supported. Because the resin I used creates relatively rigid and unbendable structures, increasing the length does not significantly impact these tools' shapes.

3.5.2 Case 2: Keyhole Pattern

In this design case, I test the insert mode to examine different material and tip lengths. The pattern I used has a "keyhole" shape that consists of a circle and a square. I segmented the pattern using the grid method. I tested a selection of synthetic fibers that have different stiffness and sizes. Figure 3.7 displays three different material tested. Wax thread is an unconventional material for brush-making. This wax-coated cotton thread has several interesting properties. To start, they are bendable yet relatively stiff fibers. Then, they naturally stick together and form clusters.

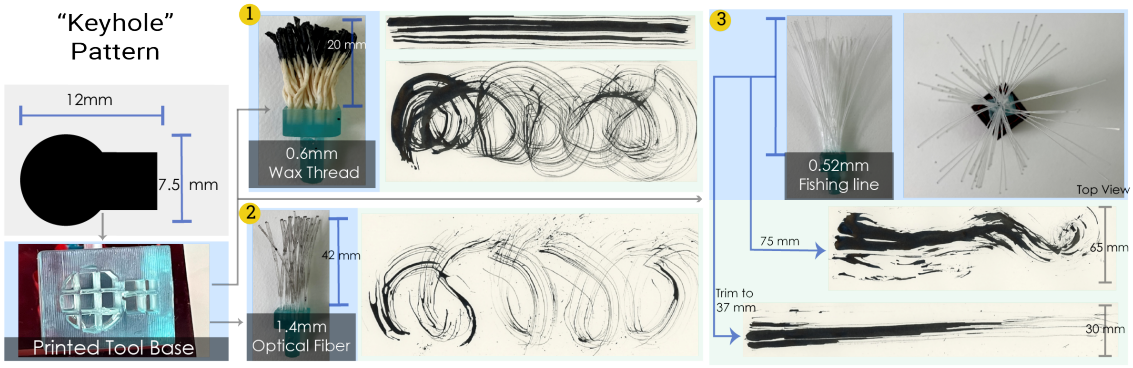

Figure 3.7: Design case 2: brushes fabricated with the insert mode using various synthetic fibers.

Also, they become softer with the increase in temperature. These characteristics create a unique mark-making experience and lead to exciting marks.

I also tested optical fibers, which are stiff plastic rods. The 1.4mm optical fiber creates dramatically different marks among various fiber thicknesses compared to the marks made with the wax thread brush. In comparison, the optical fiber brush creates a less-spread-out pattern and holds less ink, despite having a longer tip length (42mm vs. 20mm).

Nylon fishing lines are the third type of material that I tested. They are durable fibers that are stiffer than wax thread but softer than optical fibers. While fishing lines are uncommon among commercially available brushes, many brushes use nylon fibers. Figure 3.7 displays an example created with 0.52mm fishing line. After the fiber insertion process, the tips are spread out because the fishing lines are originally rolled around a rack. Users can potentially shape these tips under a heated condition, but I did not apply any treatment that adjusts the tip shape. As a result, the

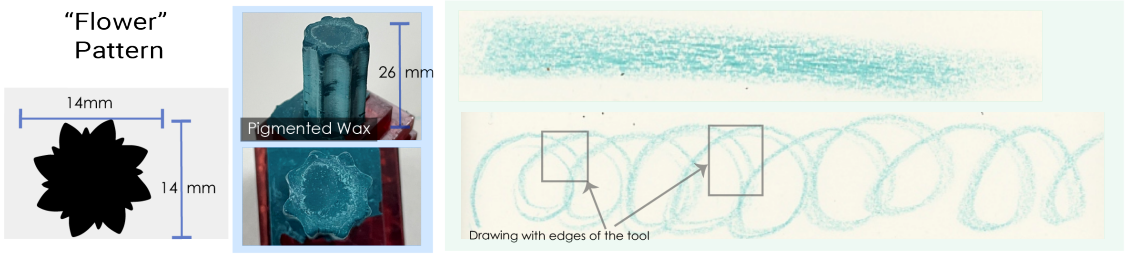

Figure 3.8: Design case 3: a crayon fabricated with the cast mode.

brush created a spread-out pattern that has exciting details. This brush is originally fabricated with fishing lines that are 75mm long. I trimmed the tips in half (37mm) and repainted samples to demonstrate the difference between tools with varied tip lengths. In comparison, the marks created with the short brush are much more concentrated.

3.5.3 Case 3: Flower Pattern

In the third design case, I tested a design using the cast mode. I used a pattern that features sharp edges of two different sizes. The mini edges were less than 1mm thick and were mostly lost during the 3D printing process. I cast the tool with pigmented wax, which preserved most of the large sharp edges. Figure 3.8 displays the side view and top view of the finished tool. Because crayon does not create dense and clear marks in one pass, I needed to repeatedly press the tool on the surface to leave a dense impression. Therefore, the pattern of the tool is mostly lost in the straight-line test. In the scribbling test, I made marks with the tip's edge instead of the top of the tip. The edge design makes it possible to create parallel lines in one stroke. Overall, the casting process was smooth. Users can reuse the printed molds

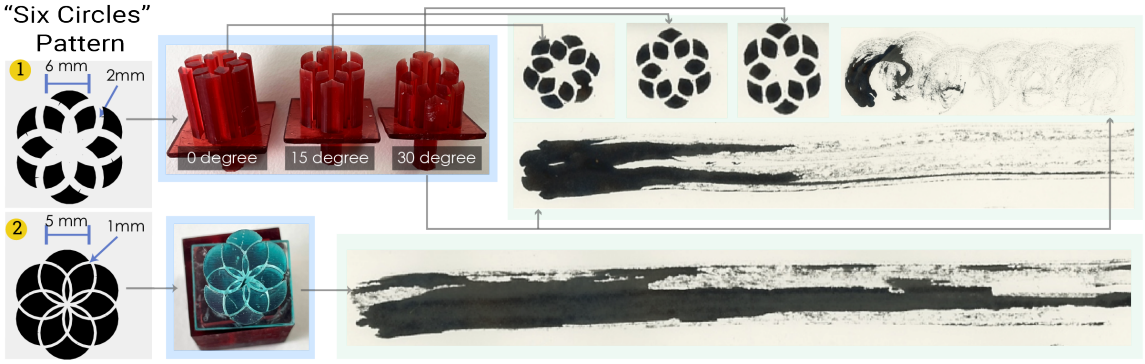

Figure 3.9: Design case 4: the "six circles" pattern is created using a simple algorithm. Creating patterns using a generative design approach enables quick experimentation towards different settings.

for multiple casts.

3.5.4 Case 4: Six Circles

In this design case, I examine patterns that are generated using an algorithm. I refer to this pattern as the "Six Circles" pattern. I generate this pattern by drawing six circles, rotating them, and remove all overlapping edges. Making patterns by defining the generative process rather than defining the output makes it easy to test different pattern settings. Figure 3.9 displays two designs (5mm vs. 6mm) with different circle and edge sizes. The smaller tool (5mm with 1mm edge) creates more condensed marks than marks made with the larger tool.

Besides varying the pattern design, I also tested this design with different tip angles. Tip angles are created by slicing the tip extrusion. As a result, tools with varying angles of tip have different touching surfaces.

Figure 3.10: Design case 5: three stamp-like tools generated with manually painted patterns.

3.5.5 Case 5: Importing Drawings

The last design case consists of three tools made by importing raster images. I created these two-color drawings using an iPad and an Apple pencil. After converting, scaling, and simplifying these patterns using *MarkMakerSquare*, I fabricated these stamp-like tools. Figure 3.10 displays the painting/stamping samples of these tools. The marks made by stamping the tool on the surface preserved these drawings well, demonstrating that *MarkMakerSquare* offers a viable solution for making stamps-like tools.

To summarize, I present five groups of instruments designed using *MarkMakerSquare*. I think these instruments and the collection of marks that they enable provide good coverage of the features that *MarkMakerSquare* currently supports. These examples also present how the six key design factors summarized in Section 3.3 impact the functionalities of mark-making tools.

Although it is potentially possible to use *MarkMakerSquare* to replicate commonly available mark-making tools, *MarkMakerSquare* is more suitable for designing

unconventional tools. *MarkMakerSquare* enables the creation of highly specialized and customized tools. Tools presented in these showcases are dramatically different from commonly available tools in terms of shape and material. For instance, a Fibonacci-patterned dip pen is unlikely to be mass-produced because it has a highly specialized purpose: making Fibonacci-related marks. The Fibonacci-patterned pen is far less versatile compared to a general dip pen. Nevertheless, its unique characteristics make it excels at creating a specific range of marks.

3.6 Future Directions

Having a toolkit dedicated to the design and fabrication of creative mark-making tools can facilitate innovative experiments and spark creative applications in the physical art & craft tool design domain. In the future, I hope to enhance this work from three major directions that include conducting additional material testing, gathering user feedback, and providing simulation support.

3.6.1 Conducting Tests in More Art-Making Scenarios

Currently, I conducted all tool tests using one art-making scenario: users are painting on a 2D surface using water-based ink. While this scenario is among the most common art-making scenarios that visual artists have, the tools I designed can perform entirely differently under other scenarios. For example, making marks with calligraphy ink is a drastically different experience than moving thick oil or acrylic paint. While I did not observe any difference made by the varying length in design case 1 (Figure 3.6), painting with acrylic instead of ink might display more salient

differences. Similarly, some of the tools I examined might create more exciting marks when they are paired with different painting techniques. Currently, I only tested addictive mark-making, which adds paint to the surface. In the future, I could test how these tools function when they are used with deductive paintings, where they remove paint from surfaces.

I only tested the usage of these tools on flat surfaces, whereas these tools might function differently when used on non-flat surfaces. For example, creating marks on pottery pieces can highlight the differences caused by varying holding angles in design case 4 (Figure 3.9). Using examples displayed in case 5 (Figure 3.10) as shapers in clay sculpting could create exciting mark-making experiences.

3.6.2 Improving Interface by Gathering User Feedback

In the future, I plan to gather more user feedback, especially towards the web-based interface that *MarkMakerSquare* is generating. Due to the challenges imposed by COVID-19, I am currently unable to host in-person design workshops where users can share insights regarding their interaction with the tool. I plan to open-source *MarkMakerSquare* and gather user feedback online. By making the toolkit publicly available, I plan to follow up with users who design, construct and use their designs outside a lab setting.

3.6.3 Providing Digital Mark Simulation

The current *MarkMakerSquare* provides an adjustable 3D model of users' design. Based on the simulation, users can make predictions of marks that a printed tool can

potentially make. For example, seeing the Fibonacci (Figure 3.6), users can roughly guess marks that the tool can create (i.e., rectangular streaks). Nevertheless, it is much harder to predict marks for tools made with the insert mode and the cast mode. For instance, design case 2 (Figure 3.7) demonstrates how fiber choice leads to a drastically different range of marks. Similarly, if I were to fabricate the flower pattern brush (Figure 3.8) with silicone instead of wax, the marks I could create with the silicone tool are likely to be completely different from the wax tool.

I currently did not provide a digital simulation for the range of marks that a tool can create because of challenges associated with vast material options and various painting techniques. To the best of our knowledge, there is no stroke-simulating model that can capture the complex range of make-making tools that *MarkMaker-Square* supports. Nevertheless, offering such simulation would effectively connect digital prototyping practices with the physical building so that users can experiment and modify their designs digitally before committing to a final plan. With the rapid development of digital simulation research, I plan to enhance *MarkMakerSquare* by adding a digital simulation that predicts the potential marks that a tool can create.

3.7 Discussion

In this section, I reflect on challenges that I encountered when designing and testing *MarkMakerSquare*. They may be helpful to those who develop design-aid systems for creative physical objects that leverage multiple fabrication methods and material. I also share high-level strategies that I took to tackle these challenges.

Many specific challenges are due to the restrictions imposed by fabrication meth-

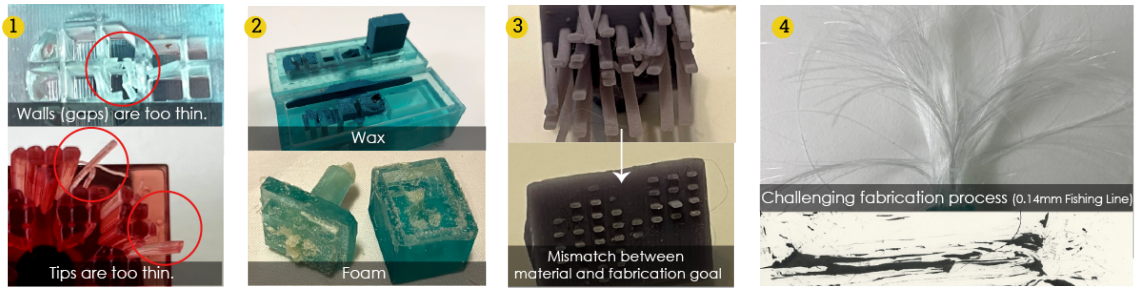

Figure 3.11: Four categories of challenges that I encountered when developing and testing *MarkMakerSquare*.

ods and material. For instance, while the maximum XY-axis resolution that my printer support is 0.047 mm, I cannot print detailed designs that are finer than 1.5mm. Group 1 in Figure 3.11 demonstrates two designs that have fabrication issues. Restrictions associated with fabricators are sometimes hidden knowledge that can only be obtained through practice. Depending on the specific material and fabricator combination, users might not be able to adopt my default settings directly. In addition to fabricator-associated challenges, different material also require users to adjust and iterate on their designs. For instance, I encountered many failed designs that are created using the cast mode. When casting with rigid foam, I discovered that it is almost impossible to release the mold, as the foam would stick to the resin mold tightly. Casting with wax is comparatively more straightforward, but releasing fine details from the mold is still tricky.

Because *MarkMakerSquare* aims to support a wide range of material and fabrication methods, it is challenging to offer a "universal" set of settings or design guidelines for all users. They might face completely different sets of challenges imposed by their fabricators and material. To tackle this issue, I provide easy access to

all parameters used for generating a design. As discussed in Section 3.4, users can input or modify settings for generating the 3D model. By making all parameters visible and editable, users can easily adjust their designs to fit their unique fabrication processes/goals.

Another issue that surfaces repeatedly is the difficulty in predicting fabrication needs. Initially, *MarkMakerSquare* only contains one fabrication mode: the print mode because I assumed that 3D printers could print brush-like tools. I could directly print brush-like structures (Figure 3.11, group 3) indeed. Nevertheless, the delicate tips of these tools gradually fall out as I make marks with these tools. This is an example where the material (resin) and the fabrication goal (make brush-like tool) are misaligned. While there are opportunities to test unconventional material-fabrication goal combinations, aligning the optimal material with the right fabrication goals could make the designing/making process much more productive.

Also, I acknowledge that there are design goals that *MarkMakerSquare* can fail to support. For instance, I tested extra-fine fishing lines using the keyhole insert design (Design Case 3) described in Section 3.5. Although it is possible to create a brush using this fine nylon fiber, the manual fiber assembly process is quite tedious. I manually bundled strands of fibers together and inserted these fiber bundles into the tip holes. However, because the fiber is too soft, I made many failed attempts before finishing a brush. In this case, using a ferrule to fasten soft fibers could be much easier than inserting these fibers into tip holes. Because it is almost impossible to address all design goals and fabrication needs, I think it is crucial to make design-aid systems extendable. *MarkMakerSquare* is fully extendable and customizable so that

users can potentially add a fourth fabrication mode that supports ferrule designs.

3.8 Conclusion

In this chapter, I examine existing design and fabrication processes for mark-making instruments. After discussing challenges in making customized creative mark-making tools, I identified six key factors that mark-making instrument designers can consider. I contribute an open-source, Python-based toolkit called *MarkMakerSquare* that supports designing and making creative mark-making instruments using different material and fabrication methods. I demonstrate a collection of instruments designed with *MarkMakerSquare* and various marks that these instruments enable. Hoping this work could facilitate innovative experiments in the physical art & craft design, I reflect on challenges encountered during the development process and share high-level strategies for tackling these challenges.

Creative goals are too diverse to predict. Although I limited the scope of this research by focusing on six factors regarding tip designs, it is still challenging to produce a tool that covers all design possibilities and considerations. Therefore, I gradually extend *MarkMakerSquare* with different fabrication modes and methods to support additional design tasks. The development process of *MarkMakerSquare* demonstrates how extendability is crucial for GCDTs. By making design software easy to extend, developers of GCDTs ensure that their tools stay relevant.

CHAPTER IV

Delicate Punch Needle Embroidery

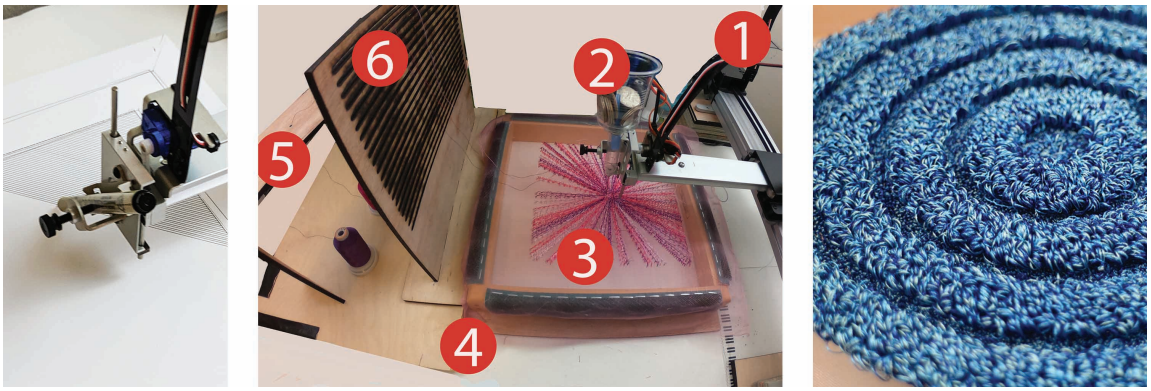

Figure 4.1: **Left:** A standard X-Y plotter that we repurposed to fabricate punch needle embroidery. **Middle:** Our modified plotter consisting of 1) an *Axidraw* plotter, 2) a customized punch needle tool, 3) a gripper frame, 4) a frame holder, 5) a threading station, and 6) a thread separator; **Right:** One example of the many styles that we can produce (3D embroidery).

4.1 Introduction

The implementation of craft designs defines the final results of these creative processes. Therefore, design-aid tools should consider the fabrication processes that

artisans use and offer support for these processes. As mentioned in Section 1.4.2, "fabrication-ready" design have different meanings under different context. Although the term "fabrication" is often associated with industrialized and machine-enabled processes, design-aid system designers should not limit their support to digital fabricator supports. For many craft, artisans might prefer to implement designs manually. There are opportunities to create design software for non-computer-controlled fabrication processes.

This chapter provides an example where design-aid tools can support manual construction by generating designs of intermediate results. In this section, I present a GCDT designed for punch needle embroidery. The GCDT called *ThreadPlotter* supports the design and fabrication of plotter-based delicate punch needle embroideries. Because the fabrication method is previously unexplored, this chapter examines a large combination of material and hardware options. As a result, the *ThreadPlotter* contains a hardware solution in addition to design software.

Punch needle embroidery is a traditional embroidery method where loops of threads are punched into backing fabrics using a tubular needle (Figure 4.2). As with other embroideries, punch needle pieces can function as decoration and textile art. However, punch needle embroidery is also commonly seen in rug production because of the unique textures it can create (*Oxford* (2016)).

There are a variety of punch needle embroidery tools. Manual punch needles are the most popular and accessible options (Figure 4.3). While the designs of these tools vary, they mostly feature two main parts: a tubular, sharp-head needle with a threading eye and a handle to attach the needle. For rug making and large-scale

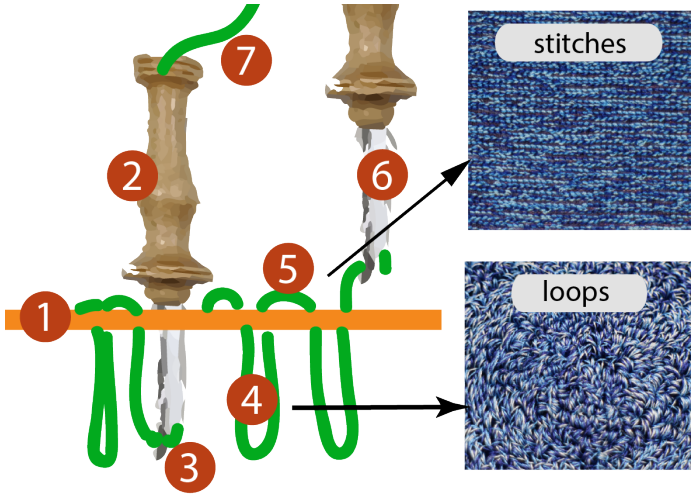

Figure 4.2: Materials and mechanics of punch needle embroidery: 1) the backing fabric, 2) a punch needle tool with two parts: handle and needle (also called "head"), 3) a punch needle tool punches through the backing fabric to make a thread loop, 4) loops created by the punch movement stay on the other side of the fabric, which is typically considered as the front side of the embroidery, 5) the thread connecting adjacent loops forms stitches, 6) a punch needle tool normally punches away from the previous loop to avoid damaging threads.

pieces, thick yarns are paired with large-sized needles. In contrast, *delicate punch needle embroideries* are made with thin embroidery floss (see Figure 4.3 for commonly used threads). These delicate embroideries are called "miniature punch needle embroidery," regardless of the dimensions of the finished pieces (*Stewart* (2009)). Mechanically, punch needle embroidery only requires three types of movement (Figure 4.2): a threaded needle punches through the backing fabric; the needle is pulled out, leaving a loop of "unpulled" thread underneath the backing fabric; and finally, the needle is moved to the next position (*Oxford* (2016); *Stewart* (2009)). Despite its simplicity, punch needle embroidery is "fragile" as it depends on the friction between fabric and thread to hold the material in place. Thus, delicate punch needle

Figure 4.3: Commercially available manual punch needle tools and commonly used threads: 1-2) Oxford Punch Needle® and Ultra Punch® Needle, 3) a variety of punch needle heads and their interchangeable handles, 4-9) examples of threads with diverse thicknesses and material.

embroidery is a labor-intensive and time-consuming craft that may not be as easily automated as other styles. Automated solutions tend to be either industrial tools or solutions that only partially automate this embroidery practice. The former include heavy-duty rug-making systems that work with thick material. The latter include tufting guns – handheld electrical tools that accelerate punch needle embroidery production by executing the punching and pulling actions with motors. While a tufting gun dramatically increases the embroidery speed, practitioners still need to "drive" the power-tool-like tufting gun manually to create desired patterns. The cost of these machines and the limited material they work with (heavy yarn) make them impractical for more delicate work. Delicate punch needle embroidery is present across a diverse set of cultures but is always done manually. Examples include the fine handheld needles such as the Russian Igolochkoy™ punch needle and the Japanese Bunka

needle (*Stewart* (2009)).

Because of the considerable manual labor involved and the lack of economical digital tools, *delicate* punch needle embroideries are often *small* in size. Large embroidery pieces require significant labor investment in addition to the material costs. Because they are often hard to produce, delicate punch needle embroideries are not necessarily attractive as design or rapid prototyping mediums, which is unfortunate, as they are texturally interesting and potentially adaptable for wearable (e.g., *McCann and Bryson* (2009)) and embroidery-related research (e.g., *Tsolis et al.* (2014)).

I demonstrate how a low-cost X-Y plotter can be repurposed into a delicate punch needle embroidery fabricator to address this challenge. By adding easy-to-make physical accessories and a software toolkit, I support the production of delicate punch needle embroideries in a precise and efficient fashion. I demonstrate how this novel and accessible fabrication approach enables the production of various artifacts and textures.

I contribute analysis (and solution) to the challenge of converting a plotter into a delicate punch needle embroidery fabricator. Additionally, I identify the specific constraints of automated punch needle fabrication where hardware, software, and material interact uniquely. I describe an open-source toolkit, *ThreadPlotter*, that supports the designing, editing, and printing of images as punch needle embroidery.[1] Finally, I reflect on how crafting experience and practice with 1) the manual form of punch needle and 2) the automated form of plotting translate (or not) to new fabrication technologies.

[1] *ThreadPlotter* is available at `http://eyesofpanda.com/projects/thread_plotter`.

4.2 Related Work

4.2.1 The Fabrication of Punch Needle Embroidery

Although the origin of punch needle embroidery is unknown (*Stewart* (2009)), this versatile craft technique is widely used across many different applications. Rug-making is likely the most well-known. In the late 1800s, a variety of punch needle tools designed for the making of "New England Style" rug hookings became available in the United States (*Oxford* (2016)). However, punch needle embroidery is neither fixed geographically nor in how it is applied. For example, Igolochkoy embroidery (also called Russian punch needle embroidery) and Japanese Bunka embroidery utilize thin embroidery floss to create much more delicate patterns (*Stewart* (2009)). Both approaches share a similar type of fine needle but use different types of thread. The Igolochkoy embroidery is found as decorations for the Russian Old Believers' traditional costumes, whereas Bunka embroidery utilizes special curly rayon thread to create textured wall hangings (*Stewart* (2009)).

Punch needle embroidery excels at creating textures because of several defining characteristics:

1. In contrast to embroidery techniques that tie the thread to the fabric, punch needle *does not secure threads to the fabric* – allowing for faster motion. The tension within the fabric holds the threads in place (Figure 4.2). The simple punch-pull movement makes punch needle embroidery an easy technique for beginners and professionals alike.
2. Punch needle embroidery is a "backward" embroidery. Instead of working from

the front side, a horizontally reversed image is *punched from the backside of the fabric*. Two sides of a punch needle embroidery piece have drastically different textures (Figure 4.2). Typically, the side with thread loops is the front side of the embroidery. The flat stitches are the backside of the embroidery (though some techniques reverse this).

3. Punch needle embroidery is a *3D embroidery technique* where practitioners can easily incorporate depth into the design. Punch needles typically come with gauges (also called stoppers) that fix the loop length by limiting the depth that the needle can be punched into the fabric. Practitioners change the gauge location to adjust loop length, thereby creating thick or thin embroideries. The use of thread trimming can create additional 3D forms.

Fully automatic industrial tufting machines make it possible to produce punch needle embroidery rugs at a speed and precision that handheld tools cannot compete with. However, automatic tufting machines are generally inaccessible to most punch needle practitioners because they are often large and expensive. Examples range from the AutoTuft to the Mtuft machines (with prices ranging from $15k to $1.5M USD). Moreover, these automatic machines and more affordable handheld tufting guns are primarily designed for rug production. Therefore, they tend to support thick rug yarn only. My goal is to produce an approachable, automated solution that maintains the advantages and uniqueness of delicate punch needle embroidery.

4.2.2 Repurposing Fabricators

With the rise of the maker culture (*Tanenbaum et al.* (2013)) and advances in fabrication research, I came across various projects that develop custom CNC machines for specific fabrication projects. Laser cutters, 3D printers, and CNC mills have become increasingly available to hobbyists and individuals (*Willis* (2018)). As the material and technology for CNC machines mature, designers and researchers have also developed specialized CNC machines to support the fabrication of craft objects. For example, *Hudson* (2014) demonstrates how a customized 3-axis machine can print needle felting sculpture. Others have found ways of adapting existing fabricators (e.g., salt and coffee-based 3D printers shown in *Rael and San Fratello* (2018)). Given the ability to customize computerized fabricators, unconventional material such as food can now be digitally enhanced (*Schoning et al.* (2012)).

Developing and assembling 3-axis CNC machines has become significantly easier as many open-source projects become available (e.g., the Maslow CNC). Nevertheless, producing a functioning and precise CNC machine remains to be a technically challenging and labor-intensive task. Instead of designing a new machine from scratch, I aim to convert an existing machine. Doing so allowed us to focus on finding the fundamental fabrication requirements rather than solving machine-specific design issues. I demonstrate how a broad type of machine intended for one task can fabricate delicate punch needle embroidery. More practically, repurposing existing machines can relieve the technical challenges of developing and calibrating physical and software components. If existing machines can be repurposed to produce delicate punch needle embroidery at a desirable quality and efficiency, it would likely make the fab-

ricator more accessible, as users might already be familiar with the hardware and the software that come with the machine. In this chapter, I aim to impose minimal physical changes to the existing machines and ensure that the machine can still operate for its original functions.

4.2.3 Applications of Punch Needle Embroidery

A specific motivation for this chapter was to enable novel applications of punch needle embroidery. In addition to rug making and fabric decoration, I have also seen punch needle embroidery being used as a way to produce customized fabrics. These fabrics can be further processed into decorative and functioning artifacts. There are examples of using this unique texture in customized plush toys, mini 3D floral sculptures, furniture covers, and bedding covers (*Oxford* (2016); *Stewart* (2009, 2013)).

With recent developments in personal fabrication (e.g., *Mota* (2011)) wearable technology (e.g., *McCann and Bryson* (2009)), and algorithmic craft (e.g., *Jacobs* (2013)), there are many craft-based research projects that explore the design and application of traditional fiber crafts (e.g., *Frankjær and Dalsgaard* (2018)). An entry-level sewing and embroidery machine makes it possible for designers and researchers to develop and test the possibility of using traditional embroidery to embedded electronics (*Hamdan et al.* (2018); *Nabil et al.* (2019)). Accessible knitting machines enable a vibrant group of studies that specializes in the design, simulation, and execution of knitting patterns (e.g., *Igarashi et al.* (2008); *Leaf et al.* (2018); *Wu et al.* (2019, 2018)). In addition to technologies that are related to the fabrication

process, the existence of efficient knitting fabricators makes it possible to utilize knitted artifacts as mediums for wearable and sensing studies such as *Guo et al.* (2011) and *Raji et al.* (2019).

Compared to other thread-related crafts such as traditional embroidery and knitting, punch needle embroidery pieces are less likely to be used as design and prototyping material despite their unique textures. An accessible automated punch needle embroidery fabricator can enable a greater variety of punch needle embroidery applications.

4.3 Physical Setup

A standard punch needle embroidery set-up requires fabric, fabric stretcher, thread, and needles. As any practitioner would attest to, any specific choice of one element will restrict (and inform) the choices of the others. To these elements, I must factor in the constraints of the mechanical fabricator–the plotter.

4.3.1 Selecting the Right X-Y Plotter

X-Y plotters are computer numerical control (CNC) machines that guide plotting tools (such as pens and markers) along vector paths. Although the name "X-Y plotter" might suggest a 2-axis machine, they are often movable along a third axis to allow the pen to move off the drawing surface. This lift provides for z-axis movement. Today, X-Y pen plotters are accessible machines. Models range from consumer-friendly self-assembling kits such as mini processor-powered Makeblock® robot kit to heavy-duty HP® vintage plotters.

3-axis movement is a minimum criterion for a punch needle fabricator. Other basic criteria include:

1. *The distance between the plotting tool and the plotting surface is adjustable and sufficient for holding a fabric stretching frame.* Different types of plotters control the x– and y-axis movement with different mechanisms. For example, a movable arm can control one or both axes. Here, the plotting surface is fixed while the arm travels. Alternatively, the plotting tool itself might be fixed while the plotting surface travels. While both approaches are viable for punch needle embroidery production, the later design might have less flexibility in the distance between the plotting tool and the plotting surface. For example, the HP7550 plotter, which utilizes a paper-feeder to control the y–axis movement, cannot hold a fabric stretcher without significant modification.
2. *The z–axis movement is large enough to create a minimal stitch.* The distance traveled in the z direction controls the size of the thread loop. Furthermore, if z–axis movement is controllable, I can fabricate punch needle embroideries with various loop sizes.
3. *Sufficient downward force can be applied in the z–axis to punch through the tightened fabric.* The force required to punch through the backing fabric varies due to fabric thickness, needle size, and stitch density. Some X-Y plotters do not provide any downward force along the z–axis at all. Instead, they rely on gravity to lower the plotting tool and only sufficient upward force to counteract lightweights. While I can add weight to the drawing tool, the mechanism to raise it may no longer work. It is also worth noting that some plotters are

designed to plot on a non-horizontal surface. Therefore the weight-adding approach might not be applicable to these machines.

Based on the criteria above, I chose to convert the commercially available *AxiDraw* Pen Plotter. The AxiDraw is a high-precision 1-arm X-Y plotter designed for plotting on flat surfaces. It utilizes step motors to control the movement along the x– and y–axis. It does not provide a downward force along the z–axis but utilizes gravity to lower the pen (or needle). In addition to meeting my technical requirements, the AxiDraw is also economical and has accessible operating software. It provides two control interfaces: an Inkscape plugin where end-users can import and plot vector graph and a Python API to control the machine programmatically. Various AxiDraw models support different plotting sizes. For my work, I used the V3/A3 model that comes with a plotting area of 11×17 inches.

4.3.2 Fabric and Fabric Stretcher

The backing fabric is an essential part of the punch needle embroidery because the tension between the weaves of the fabric is the only thing that holds the thread loops in place. The fabric and thread need to be matched. For example, a loose-weave fabric will not provide enough force to hold thin thread but might work with thicker thread. Conversely, punching through a tight-weave fabric with a large needle (for a thick thread) will require significant force and likely damage the weave.

Thick yarn punch needle embroidery is mostly done on monk's cloth, an even-weave cotton fabric that contains tiny holes formed by the warp and weft threads (*Oxford* (2016)). When making delicate punch needle embroidery, the most popular

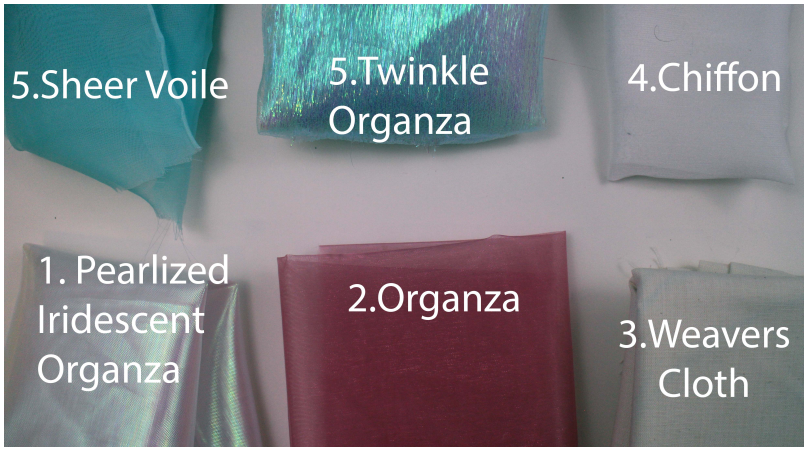

Figure 4.4: I tested six different types of fabrics. Pearlized Iridescent Organza (1) and Organza (2) can produce stable results. Weavers Cloth (3) is too thick for my machine to pierce. Chiffon (4) and Twinkle Organza (5) are too fragile for the gripper frame I used. Sheer Voile (6) can produce reasonable results but tends to have missing loops.

fabric is weavers cloth, a polyester-cotton blend fabric. In addition to weavers cloth, a variety of fabrics such as muslin, cotton chambray, wool flannel, silk noil, and linen might also work with particular combinations of needle and thread (*Stewart* (2009)).

The choice of plotter constrains my choice of fabric. For example, the AxiDraw does not provide enough downward force to punch through most fabrics. This issue can be partially solved by increasing the weight of the punch needle. However, in my experiments, I discovered that increasing this weight too much would quickly wear out the servo motor controlling the pen lifter. In experimenting with various fabrics, I found that thin fabrics, such as organza and voile, were pierceable without much change to the stock AxiDraw. Additional modifications (e.g., a heavy-duty servo on the pen lifter) would allow for thicker fabrics.

In addition to fabric choice, I found that the choice of fabric stretching mecha-

nisms was critical for smooth operations. Unlike traditional embroidery, where the stretching of the fabric might be optional, punch needle embroidery requires tightly stretched fabric. It is crucial to stretch the fabric "drum-tight" because loosely stretched fabric requires considerable piercing force (*Oxford* (2016)). Stretching the fabric also reduces the damage to the fabric during the punching process. A vari-

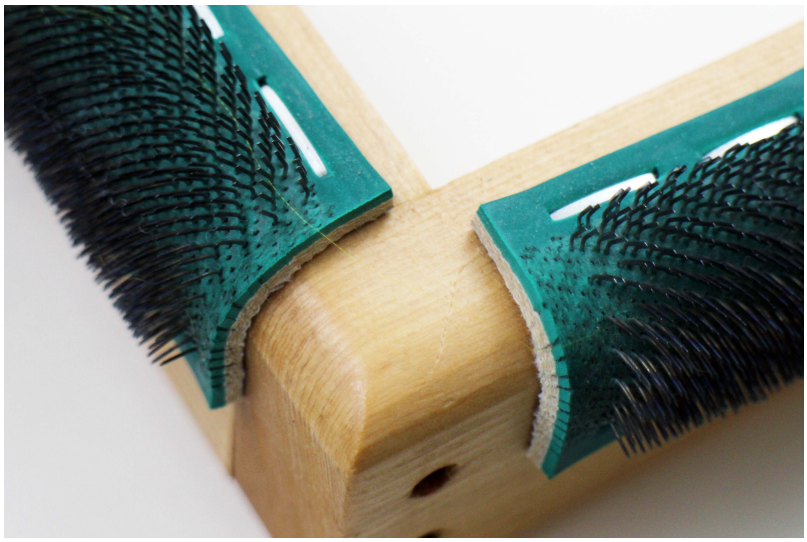

Figure 4.5: I recommend using gripper frames for plotter embroidery. They secure and stretch fabrics with curved metal needles that grasp the fabric.

ety of embroidery hoops and gripper frames can be used for manual punch needle embroidery. Embroidery hoops are circular stretchers that secure and tighten the fabric by clasping the fabric between the inner hoop and the outer hoop. They are economical, lightweight, and adjustable stretchers that would work for manual punch needle embroideries. However, the embroidery hoops I tested could not stretch the fabric to be tight enough for machine-based embroidery. A loose stretch that may work for manual crafting will fail when I use the more delicate servo motors of the

plotter. Additionally, the unevenly stretched fabric will also cause uneven piercing, impacting both loop height and density.

Given this, I found gripper frames to be ideal for plotter-based punch needle embroideries. Gripper frames are non-adjustable solid frames covered with metal gripper strips that are made with bent metal needles (called "teeth") (Figure 4.5). Gripper frames are more secure than general embroidery hoops because these sharp needles prevent any slipping. It is important to note, however, that it is possible to overstretch some fabrics. Large teeth might also tear the delicate fabric. It is crucial to pair the right fabric with the right teeth size and avoid overstretching. In my experiments, I used a 10×10 inch wooden frame covered with EH4 gripper strips manufactured by Howard Brush.

Finally, I found that it is critical to secure the frame to a stable surface to prevent unintentional movement. Even a heavy frame may move enough during the fabrication process to ruin a piece. To prevent this, I designed a simple wooden holder to secure the frame. The holder also works as a registration tool that ensures I place the frame at the same location every time (Figure 4.1).

Figure 4.4 displays a variety of material that I have tested–with organza being my primary choice. Voile, cotton, and chiffon might also be viable options if paired with the right thread and gripper teeth size. All examples shown in this chapter were fabricated with organza.

4.3.3 Thread and Thread Feeder

Theoretically, any thread that "flows easily through the needle and leaves even, consistent loops in the fabric will work (*Stewart* (2009))." For delicate punch needle embroidery, the most frequently used thread is cotton embroidery floss. Embroidery flosses are widely available threads manufactured for embroidery-making that come in a variety of colors and fibers. Six-strand pre-cut cotton embroidery floss is one of the most popular embroidery threads (*Stewart* (2009)). However, pre-cut embroidery thread requires practitioners to re-thread the needle quite frequently. Therefore, continuous spools are preferable.

I tested wool, cotton, polyester embroidery thread, and polyester metallic thread (Figure 4.3). I identified the following three factors to be considered when selecting threads:

1. **Thread thickness**: Pairing fine thread with loosely-weaved fabric cannot work because the tension within the fabric weave cannot hold the thread loops. Pairing thick thread with fine needles cannot work either because the thread cannot pass through the needle freely. Additionally, a larger needle requires more piercing force, and the piercing force provided by my plotter is limited by the maximum weight the plotter pen lifter can hold.
2. **Thread Smoothness**: In order to form thread loops, the thread needs to flow freely through the needle. Natural fiber such as cotton and wool thread might come with tiny strands of fiber that increase the friction. Additionally, it is common to use multiple strands of threads to increase thread thickness and to blend color. In these cases, natural fibers can tangle if they have fuzzy finishes.

Similarly, metallic threads that tangle easily might not work with plotter-based punch needle embroidery.

3. **Thread strength**: When making manual punch needle embroidery, the punch needle is angled so that it always punches away from the thread, ensuring that the sharp head of the needle does not damage the thread. X-Y plotters are generally not equipped with a pen rotation mechanism. Varying holding angle dynamically is rarely needed for general plotting activities, so most plotters fix the pen angle. The fixed angle might lead to cuts in weaker threads. To avoid this problem, I choose to use strong threads that are less likely to break or become fuzzy even when pressed by the punch needle.

Among the threads tested, polyester embroidery threads produced the most stable results. Polyester embroidery threads are strong and smooth synthetic threads that are manufactured for machine embroidery. A variety of sizes and colors are available. I used 120 deniers, two-ply 100% polyester embroidery threads that are widely available on the market. I tested punching one, two, and three strands of this embroidery thread through organza. Three strands of thread produce the most stable result.

A thread feeder is necessary because I rely on continuous thread spools. Standard embroidery machines usually have a thread tension adjustment system to ensure the thread is tightened. In contrast, punch needle embroidery requires that there exists no tension on the thread. Even the slightest tension would cause the thread loops to pull out of the fabric along with the punch needle. Fortunately, most embroidery spools are designed to unwind from the top. As long as the force to unwind the

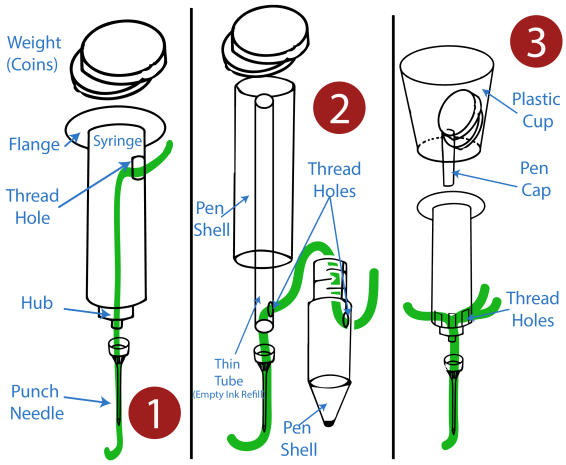

Figure 4.6: Three iterations of the punch needle handle design using pre-manufactured plastic material. Through these prototyping iterations, I locate the optimal material (i.e., syringe), location for the threading holes (i.e., close to the syringe hub), and mechanism to add adjustable weight (i.e., use the syringe flange to hold a plastic cup).

thread is directly above the thread, it is effortless to unwind. To ensure that the force unwinding the thread is directly above the spool, I designed a simple thread unwind station where individual spools are pulled and passed to the punch needle.

4.3.4 Punch Needle

The last physical component I designed is the punch needle. For plotters that provide piercing force, it is theoretically possible to use any commercially available punch needles. However, the handles on these needles are often difficult to modify (e.g., for adding weight). All punch needles I examined accept thread at the end of the handle, making it impossible to add weight directly above the needle (see the threaded tools in Figure 4.2 and Figure 4.3). Adding weight to the side of the handle or the pen lifter is less effective.

In addition to the weight issue, some fine punch needles have very short handles that are not long enough for AxiDraw to hold. Because of these constraints, I designed my own punch needle handle that can: 1) holds commercially-available punch needle heads, 2) holds an adjustable amount of weight directly above the needle, and 3) allows for thread feeding without tangles.

To make my design as accessible as possible, I searched for economical material that can be easily converted into the handle. I selected pre-manufactured plastic and tubular material (e.g., syringes and pen casings) as the primary material for rapid prototyping. These were cheap, easily available, and could be modified without specialized equipment. I designed and tested three different handles using syringes, plastic pens, coins (as the weight), and plastic cups. In my progress towards my final design, I identified several key constraints.

First, I need to make the threading process as simple as possible. In existing handle designs, threads are normally fed into the needle using metal wire threaders. However, straight threaders are less effective when the threading pass is not straight. In the first design iteration, I made a curve threader to pass the thread from the side. Nevertheless, operating a curved threader requires end-users to aim for the needle, which complicates the threading process significantly. In the second iteration, I place a tubular insert made from an empty gel pen lead inside of a 2-part twist pen. Threading the gel pen lead is easier than using the curved threader. However, because the diameter of the insert is small, threading multiple strands is still a time-consuming activity that also requires a fine crochet needle to pull the thread.

As a result, I separated the threading into two parts in the third iteration. I

placed the threading holes close to the hub, which makes it possible to pass the thread through the hub without a threader. After passing the thread through the hub, I use a straight wire threader to thread the needle. Then, I attach the needle to the syringe. Besides adjusting the location of the threading hole, I also noticed that multiple thread holes are necessary if multiple strands of threads are used. Feeding individual strands of thread to dedicated thread holes reduces the chance of thread tangling dramatically. Consequently, I designed three threading holes on the barrel of the syringe.

Second, I needed to design a platform to hold adjustable weight. I chose to use coins as weight in my prototypes because they are accessible "heavy" metals. In the first iteration, I taped coins to the syringe flange. This worked well with the caveat that it was difficult to adjust the number of coins. In the second iteration, where I re-purposed a pen, I laser cut a flat wooden square and taped coins to it.

I noticed that the syringe flange and hub are handy structures for the punch needle handle when comparing the first two iterations. The syringe hub fits most of the fine needles I examined. Therefore it automatically holds the needles vertically. The flange of the syringe is flat, making it easy to add weight on top. To make a platform that can hold an adjustable number of coins, I combined a cylinder-shaped pen cap and a small plastic cup. The glued-together platform sits directly on top of the flange. End-users can place any number of coins inside the cup. The cylinder-shaped pen cap ensures that the platform stays on top of the syringe without the need for adhesives.

The final punch needle design is economical and straightforward. In comparison

to the first variants, it features easy threading processes and an effortless weight adjustment system. An optional component to add is a gauge/stopper that indicates the location where the needle should be held. End-users can use the existing volume markings on the syringe as guidelines for placing the punch needle tool. End-users can also use tapes and markers to indicate the desired punch needle location. Figure 4.6 illustrates the design of my customized punch needles.

4.4 Software Control and ThreadPlotter

The AxiDraw offers two control interfaces: an Inkscape plugin where end-users can plot vector images (in the SVG format) and a Python scripting interface. The latter provides low-level controls such as *pen_up* and *pen_up_speed*. The graphical SVG interface can be used for many punch needle embroidery fabrications. However, I found that having access to low-level controls opens up more fabrication possibilities. Some design patterns that involve different loop and stitch sizes have complex setting changes. This can be "hacked" by creating various layers and fabricating them one at a time. However, because not all controls can be adjusted by inputting the images, the end-user must pause the machine to adjust the plotter settings. Access to low-level functions significantly reduces the manual labor involved.

To control my physical setup, I developed *ThreadPlotter*, a Python-based API that supports the design of X-Y plotter compatible embroidery patterns. The API primarily helps end-users address the problem: Given a vector path to be fabricated into punch needle embroidery, *where* and *how* should the machine punch? *ThreadPlotter* answers these questions by processing path information and translating con-

tinuous paths into a set of punch point locations. Additionally, *ThreadPlotter* will determine effective settings for the depth and speed of the punch. While translating vector information into vector-based punch needle embroidery patterns is the primary function of *ThreadPlotter*, the system also provides utility functions (e.g., for converting raster images to acceptable vector formats).

ThreadPlotter can produce two kinds of outputs. First, *ThreadPlotter* produces SVG files that can be fabricated through the graphical Inkscape environment. my focus on vector formats is due to the observation that these are the most widely accepted by X-Y plotters. Second, *ThreadPlotter* produces Python scripts that can be directly executed by Axidraw. While these scripts may not directly execute on other plotters, I utilize 'simple' calls that are likely to be supported by most devices. Thus, I believe *ThreadPlotter*-output scripts can be readily adapted.

4.4.1 Determining Punch Points Locations

ThreadPlotter converts vector images into punch needle embroidery patterns in three steps. To start, *ThreadPlotter* extracts vector elements such as ¡line¿ and ¡polygon¿ (in SVG syntax). For each vector element, *ThreadPlotter* identifies and processes the location information of the element. It also converts curves into straight lines in preparation for later path segmentation (Figure 4.7). For example, *ThreadPlotter* will approximate cubic Bézier curves with lines. At the end of this step, I have a list of polylines: continuous paths that consist of only straight line segments.

ThreadPlotter divides line segments within the polylines into equal-length sections. The unit length used in the dividing algorithm controls the density of the

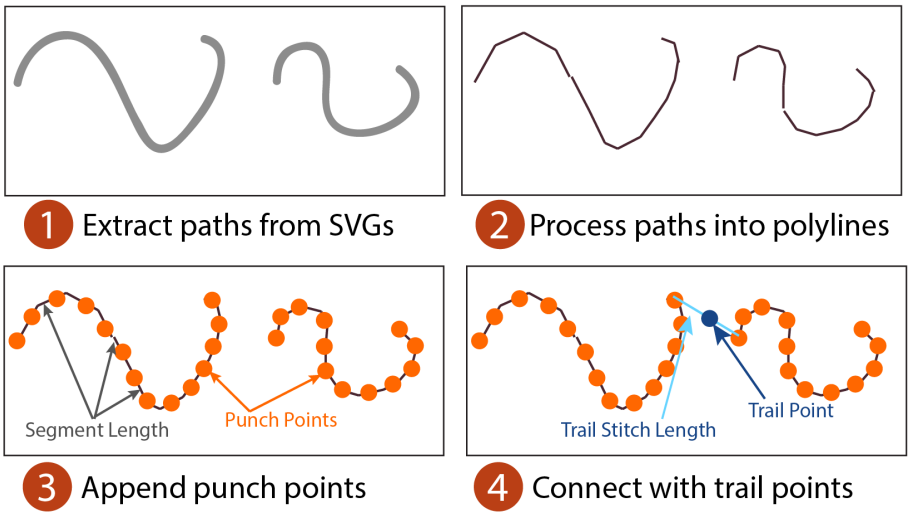

Figure 4.7: Pipeline for converting vector paths into plotter-compatible punch needle patterns.

thread loop and the size of the stitches. I refer to this length as *segment length*. Setting a suitable segment length is important for a good weave. For example, the finished piece will not have a firm, rug-like texture if the segment length is too long. If the segment length is too short, the needle will pierce a section of the fabric repetitively and damage the fabric. Different combinations of fabric and thread require different segment lengths. For my combination of fabric and thread, I found 1.05 mm to be optimal.

At the end of each line segment, *ThreadPlotter* generates a punch point. Each punch point represents a location where the needle will pierce the fabric. After annotating the end of each line segment, I obtain polylines consisting of a series of punch points. I refer to punch points within one polyline as a *punch point group*.

Finally, *ThreadPlotter* connects different punch point groups with *trail points*. Because punch needle embroidery uses a continuous, untied thread, pulling the thread

can revert the previous stitches. In other words, a threaded punch needle cannot travel a long distance without pulling the previous loops out. Due to this characteristic, a punch needle cannot directly move to the start of the next punch group after finishing one group of punch points. *ThreadPlotter* adds piercing points as supporting trails to avoid pulling previous loops.

The additional loops created on trail points ensure that the loops in the previous punch points stay in place. The segment length and loop height in the trails should be different from that of the polyline for a couple of reasons. First, as long as the punch needle touches the fabric and forms a minimal loop, the previous loop will not be pulled. Therefore, it is not necessary to punch long loops in trail points. Second, trail lines are supporting structures that need to be removed after the fabrication process because they are not part of the original pattern. Making dense trail stitches wastes material and increases the difficulty during the removing process. When punch points of the same color are scattered around the image, there might be many trail points. However, as long as the loop length of the trail points is less than that of the punch point, longer loops can cover up the shorter loops. Thus, not all trail stitches need to be removed (as they are covered up).

If the end-user opts to output an SVG file (rather than code), they can use the Inkscape interface to print. There is a tradeoff in doing so. On the one hand, the end-user can intervene to inspect and control the output properly. On the other hand, end-users must pause the plotting process to adjust the loop length (i.e., the lowest position that the needle can travel along the z–axis). An alternative to manual pausing is to separate punch point groups into different layers so that Axidraw can

stop before the transition between punch point and trail point. However, the time and labor involved in the setting adjusting process outweigh the potential benefits. To balance this, trail points in the SVG mode share the same loop length as punch points but use a longer segment length (2.64mm). In the script mode, the needle position can be adjusted automatically. Hence, trail loops have a minimal loop length of 2.2 mm. When operating using the SVG interface, the plotter will automatically return the arm to its origin. The returning path also needs to be covered with trail points to avoid pulling previous loops.

If a design has multiple colors, *ThreadPlotter* preprocesses vector paths by grouping them according to their colors. Then, *ThreadPlotter* processes individual color groups separately, generating files for each color. In both the SVG and scripted forms, some manual work is required. When users finish plotting one color, they can remove trails, re-thread the needle, and proceed to the next color.

4.4.2 Stitches, Loops, and Pen Speed

With more sophisticated embroideries, such as those with a 3D effect or the inverted Bunka stitching (where the stitch side becomes the front), the scripted output of *ThreadPlotter* shines. Without it, end-users must constantly adjust loop length and other elements. However, even the scripted interface does not offer infinite flexibility. The properties of the material and their interaction constrain what is feasible.

To understand the relationship between loop length and stitch size, I experimented with several combinations to find workable settings. I started by measuring

the maximum and minimal loop length that AxiDraw can create, then conducted several tests using different loop sizes within the range.

I found that I can approximate the relationship between loop depth (*loop_depth*), punch depth (*needle_depth* – the depth of the needle below the fabric), and stitch length (*stitch_length*) using a simple formula:

$$loop\_depth = \frac{needle\_depth - stitch\_length}{2}$$

Additionally, I observed that if the needle moves too fast, the thread might follow the needle and get pulled out of the fabric because of inertia. Short loops are especially vulnerable as they can be pulled out easily. As a result, when punching short loops, the needle needs to have a slower lifting speed compared to the speed used for long loops. With this observation, I measured the optimal needle lifting speed associated with the minimal and maximum loop length. *ThreadPlotter* can suggest the optimal needle lifting speed given the loop size using a simple linear mapping calculation.

Given the desired stitch size, I can also calculate the loop length and lifting speed that ensures the creation of a minimal loop. I used this calculation when processing trail points in the scripted mode so that I can use the maximum stitch size and the minimal loop length. Moreover, this insight enables the pattern generation for Bunka-style punch needle embroideries, where stitches of different sizes are treated as the front side of the embroidery.

4.4.3 Raster versus Vector Images

Thus far, I have focused my attention on converting a vector image into a plotter-compatible embroidery pattern. Using drafting tools such as Inkscape and Adobe Illustrator™, designers can produce precise and scalable vector images. Many tools also offer the ability to convert raster images into vector formats. *ThreadPlotter* offers direct creation and manipulation of SVG files through scripting. It also implements an alternative raster-to-vector conversion module that considers the intended target (the punch needle embroidery) in the conversion. *ThreadPlotter* uses the following conversion pipeline.

1. *ThreadPlotter* loads an image in the format of JPG or PNG, then adjusts the number of colors in the image. I experimented with multiple color-reducing algorithms, including quantization (*Gray and Neuhoff* (1998)), k-means clustering (*Likas et al.* (2003)), and grouping colors that have short Euclidean distances. All three methods produce usable results.
2. *ThreadPlotter* groups pixels into squares whose width and height equal to the segment length. This process adjusts the unit pixel size to be equal to the segment length. At the end of this step, I obtain a copy of the image that has a limited palette and a specific Pixels Per Inch (PPI) controlled by the segment length. At the center of each adjusted pixel, *ThreadPlotter* produces a punch point using a random color selected from this group of original pixels.
3. *ThreadPlotter* then links punch points that share the same color using trails. The trail generating process is identical to that within the SVG mode.

Using this conversion module, users can generate punch needle embroidery pat-

Figure 4.8: *ThreadPlotter* processes raster images into plotter-compatible embroidery patterns. Original artwork by Shiqing He (*He* (2015)).

terns directly from raster images (e.g., paintings or photos). Figure 4.8 exhibits a finished punch needle embroidery piece designed and fabricated using this approach.

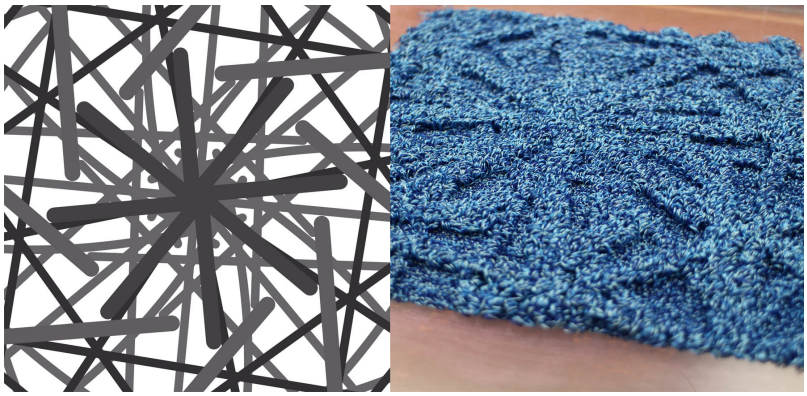

Figure 4.9: By assigning specific loop lengths to different colors, users can create patterns for 3D punch needle embroidery.

In addition to the basic pipeline described above, I built a few additional extensions to the vector production system. For example, *ThreadPlotter* can treat colors within the images as indicators for loop size. For example, it will map punch points with darker colors into longer loops. By adjusting the loop size dynamically, users

can produce 3D embroideries with one color of thread. Figure 4.9 provides a 3D example fabricated with one color.

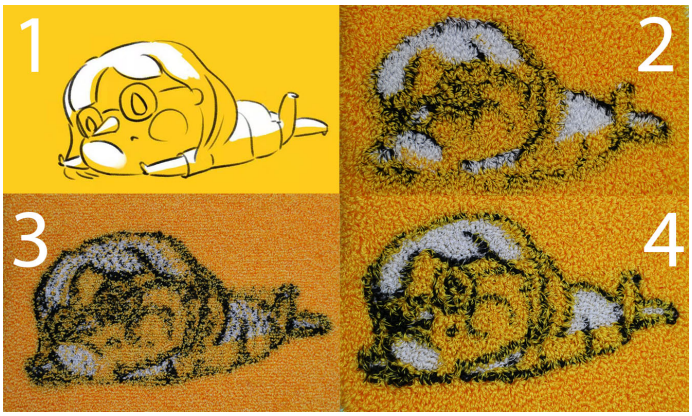

Figure 4.10: Features within images might be blurry if the embroidery size is too small. The effect is especially observable when using a long loop length (in 2, loop length = 6mm). Besides enlarging the embroidery size, users can reduce this effect by using a shorter loop length (in 3, loop length = 2.4mm), or assigning different loop lengths to specific colors (4). Original artwork courtesy of P Mei (*Mei* (2019)).

While I can convert fine details in the raster image (e.g., a thin line) to a vector, this detail may not correctly render when embroidered. Punch needle-produced loops can intertwine, making colors appear blended, and the image appears blurry. In manual punch needle embroidery, practitioners address this using sharp tools to separate these intertwine loops after all stitches are made (*Oxford* (2016)). End-users can certainly do this additional step after the machine fabrication process.

In building *ThreadPlotter*, I have identified three alternative approaches to support printing detailed images without manual intervention (Figure 4.10). The most straightforward and effective approach is to scale up the embroidery size. Scaling up will increase the "resolution" of the finished piece, therefore making the finished piece more recognizable. Users can also consider using a short loop length. When

the length of the loop is short, loops are less likely to blend. However, the shortest loops should still be longer than the trail loops. Otherwise, punch loops cannot hide trail points completely. Finally, users can assign individual colors with different loop sizes. Figure 4.10 shows an example fabricated with this approach. The piece with mixed loop length better preserves the image's original feature compared to the untreated design.

4.4.4 Thread Color Matching

Selecting the closest thread color might be a tedious task when plotting a multicolor embroidery, especially if there are multiple similar colors within the palette. The last function of *ThreadPlotter* is to provide a simple thread color matching tool. In my experiments, I gathered embroidery threads in more than sixty colors. I collected the RGB value of each thread from the official color chart provided by the manufacturer (other threads can be added). *ThreadPlotter* finds the closest color match to the color in the embroidery pattern by calculating the euclidean distance between two colors.

Because the number of shades I can gather is limited, there are cases where my closest match is still drastically different from the desired color. In these cases, I provide an experimental feature that suggests potential threads that "blend" into the desired color. The blending is possible because I can use three strands of embroidery thread in different colors for my punch needle. When looking from a distance, the distinct colors of these three strands of threads will appear as if they blend into one color. Because physical color blending is very different from digital color blending, the

blending suggestions I generated might not always function correctly in the physical world. Nevertheless, users can use these thread color suggestions as starting points for finding the ideal thread color.

I briefly summarize the overall experience of using *ThreadPlotter*. When end-users input an image (vector or raster), *ThreadPlotter* processes the image into plotter-compatible punch points and trail points. For every color within the image, *ThreadPlotter* generates an SVG file and a Python script. The SVG file contains location information of punch points and trail points. The Python script contains both the location information and machine settings, such as needle raise speed and needle position along the z–axis. Additionally, *ThreadPlotter* produces another SVG file that displays the suggested color of the thread to use. Using this toolkit, I can create a wide range of artifacts.

4.5 Discussion

Adapting an X-Y plotter – a tool intended for one type of "fabrication" – to an entirely new form, presented many challenges. While I could address a number of these challenges in my hardware and software implementation, my default settings might not entirely account for the complex interactions between material . A mistake can unravel the entire image or break the material (see Figure 4.11 for some examples). I reflect on these not only because they may be useful to those using my approach but also to describe the challenges of adapting manual techniques and expertise–specifically around troubleshooting–to the automated infrastructure.

Many specific challenges are due to the lack of human monitoring. Automation

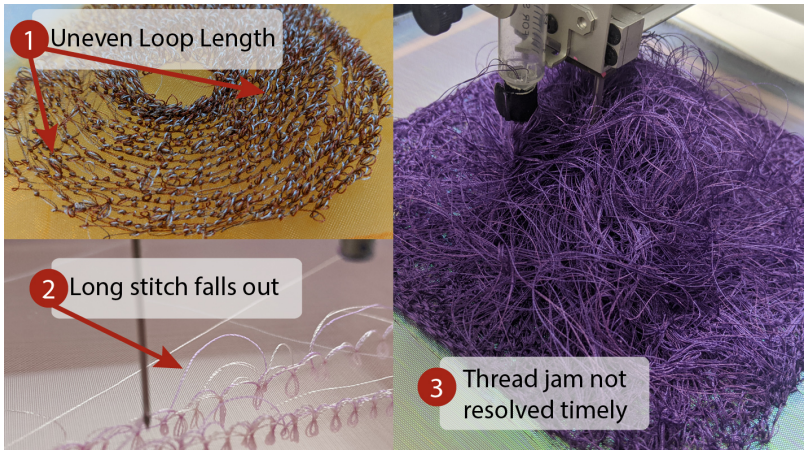

Figure 4.11: Several failed examples that demonstrate common issues.

invites one to "set-it-and-forget-it," walking away from the machine as it works. As a specific example, I found that it is necessary to be cautious when fabricating long stitches with short loop length because they are more likely to fall out of the fabric even when produced at a low speed. When loops fall out of the fabric, the extra thread hanging on the stitch side can also trigger additional tangling or even seize the moving needle.

Practitioners of manual punch needle embroidery can react, troubleshoot, and correct problems dynamically, which is only possible due to their engagement with the process and their extensive experiences with the craft. What was notable to us in building this platform were the places where these experiences could or could not help in the automated scenario.

A specific example of this relates to difficulties for the needle to pierce the fabric– a rare issue in the manual form. Reasons for this may include the obvious: a dull needle (which can be resolved with sharpening). The needle piercing problem also

includes subtle differences in how the fabric is stretched. While insufficient or uneven stretching does not present a problem for a person embroidering, this is critical for a successful automated plot. Trouble piercing the fabric can also be resolved with more "elbow grease" – simply applying more force (or, equivalently, weight). However, simply adding weight to the needle will not work if the weight exceeds AxiDraw's lifting limit.

Another issue relates to the specific problem of converting a device that fabricates one material to another. There was not always a direct translation from my understanding and experience with the X-Y plotter to the X-Y needle punch machine. For example, the AxiDraw is sufficiently heavy and robust to work with light pens. Pens tend not to "drift" up or down. While needle-punch is in some ways delicate, it invariably pushes the limits of the plotter and required adaptation. For example, I found that securing the AxiDraw down using clamps was necessary to avoid drift or shake. I also found that some plotters' arms bend when they are extended, especially when they are holding a heavy-weight needle. In this case, it is helpful to angle the gripper frame slightly to create a slanted surface parallel to the plotter arm.

In some cases, I also made trade-offs in what I supported in printing. For example, in manual embroidery, the needle should be as close to the fabric as possible. However, if a needle is caught under the fabric and it is not resolved immediately, the needle will likely tear the fabric because the plotter cannot detect this issue. Therefore, I lift my needle 3mm above the fabric. The extra distance reduces the chance of getting caught but also reduces the maximum loop length that I can fabricate.

4.5.1 Future Directions

Having an accessible punch needle embroidery fabricator can encourage innovative applications of this unique and versatile craft. The ability to fabricate customized textiles at a low cost gives practitioners the freedom to explore and experiment. Besides producing aesthetically pleasing textile art, this technique and the produced artifacts have the potential to bring a unique touch to wearable technologies and the fabrication of soft IoT devices. For example, I can imagine custom-made bathroom rugs that embed health-measuring sensors. I am also excited about the potential of using this textile as a material for building soft computing devices (e.g., *Berzowska and Bromley* (2007)). In the future, I hope to expand this study by collecting additional user feedback and analyzing the performance of the *ThreadPlotter* on other platforms such as DIY X-Y plotters. Last but not least, I hope this work spurs conversations in designing accessible craft fabricators.

4.6 Conclusion

This chapter demonstrated how a repurposed low-cost X-Y plotter could produce delicate punch needle embroideries in a precise and efficient fashion. I examined the opportunities and challenges of this novel fabrication method. Hoping to make this fabricator economical and accessible, I used easy-to-source material for building physical accessories and imposed minimal change to the plotter (ensuring that it can still be used for its original purpose). I presented *ThreadPlotter*, a toolkit that contains all physical and digital tools needed for the fabrication process. It is

publicly available at `http://eyesofpanda.com/projects/thread_plotter`. I hope this work could support unconventional applications of this versatile fiber-based craft and spurs discussion on designing accessible craft fabricators. Because plotter-based punch needle embroidery is a novel fabrication method, the GCDT developed for this craft contains a hardware part and a software part. Through this example, I demonstrate that GCDT designers should analyze potential fabrication methods associated with the craft to support users' fabrication processes.

CHAPTER V

Reflection

In this concluding chapter, I revisit the GCDT framework. The development of the GCDT framework is motivated by current challenges associated with existing design-aid systems. It is a promising way to develop design-aid software for craft disciplines that have certain characteristics (see Section 1.3.3 for a detailed discussion). By designing, developing, and testing three GCDTs in different craft domains, I demonstrate GCDTs' unique characteristics and how they support creative activities. I discuss various project-specific insights in Chapter III, Chapter II, and Chapter IV. In this chapter, I discuss overall findings and insights for developing GCDTs. I also present challenges and future directions of the GCDT framework.

5.1 Development Pipeline of GCDTs

Through developing three distinct GCDTs, I discussed opportunities and challenges associated with these individual GCDTs in the previous chapters. This section provides a summary of generalized insights associated with GCDTs development

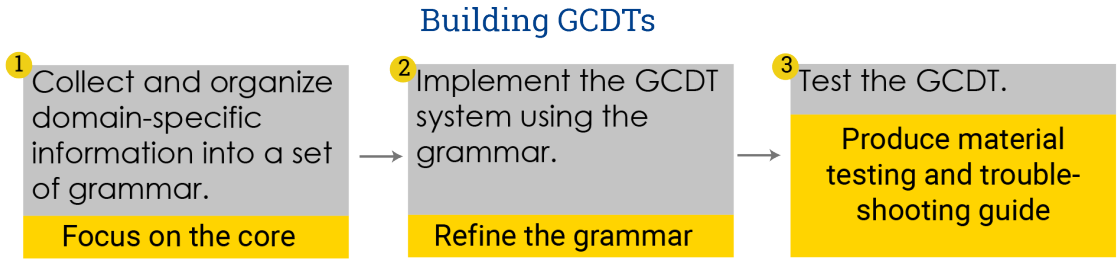

Figure 5.1: The Development Pipeline of GCDTs

processes. Overall, GCDTs are developed in three stages. Figure 5.1 summarizes this development process. To start, system designers need to collect and organize domain-specific information. Then, designers need to organize the information into a set of grammar, which constructs the language for describing and making a design. The refining process is often the most challenging step at this stage. Overall, system designers should focus on extracting core information before expanding the grammar with detailed rules and techniques. Chapter II presents an example for this insight.

The second stage of the development process is system implementation. System developers can use this opportunity to test and refine the grammar because they are likely to encounter discrepancies and inconsistencies within the initial implementation. Also, developers might discover the need for new content in grammar during this stage.

In the last stage of the GCDT development process, developers test their GCDTs. In all three examples presented in this dissertation, GCDTs are tested through examples. By showcasing various artifacts produced using GCDTs, I demonstrate the potential sets of design tasks these GCDTs can support. Also, it is essential to pro-

duce material testing reports and troubleshooting guides at this stage (see Chapter III for a detailed discussion).

5.2 Future Directions

Grammar Extraction

Collecting and organizing domain-specific information is undoubtedly the most critical challenge across all three projects. As discussed in Section 1.4, domain-specific knowledge for some craft disciplines is likely to be fuzzy, unorganized, and limited. For instance, it's possible to find textbook-like material for the art of punch needle embroidery. However, it is much more challenging to locate information for multilayer sculpture, partially because multilayer sculpture is used across many types of applications, such as installation art, lightbox design, and card design. In these cases, system designers might need to collect information in related craft (e.g., find paper cut information) or extract information by analyzing artifacts.

Besides these difficulties associated with collecting domain information, structuring domain information into sets of grammar can also be challenging. In this dissertation, I primarily focus on identifying types of information that system designers should collect. There are still many open questions regarding the extraction and refining process. Existing studies in software development and API design offer a potential collection of strategies. For example, *Vernon* (2013) discuss a pipeline for designing systems that involve domain experts. Studies like *Bloch* (2006); *Stylos and Myers* (2007); *Henning* (2009) also offer general suggestions for designing API. Currently, I take an iterative approach to organize the grammar. Chapter II

describes this process. In the future, I hope to examine the extraction processes and provide in-depth analysis for these processes.

Machine, Art, and Culture

When developing technology related to art and craft, it is critical to discuss the potential impact of these technologies. Mass production certainly changed the designing and making paradigm of many craft domains. For example, *Fisher and Botticello* (2018) discusses the production of machine-made lace and how craft knowledge changes because of machines. *Eglash et al.* (2020) discusses how mass production could damage artisanal traditions.

I believe that GCDTs should support new ways of designing and making instead of replacing manual processes. It is essential to acknowledge that machine-made artifacts can never replace handmade craft artifacts. When developing new craft technology, developers should examine how the technology broadens the design space of a craft domain rather than substitute traditional practices. For example, the development of digital cameras should not be advertised as replacements for manual cameras. Instead, digital cameras are offering new ways of making images. In this case, it is also crucial to preserve manual photography practices.

Besides examining the differences between machine-made artifacts and handmade artifacts, developers should also facilitate human-machine collaboration. Studies such as *Eglash et al.* (2020); *Devendorf* (2016) and *Albaugh et al.* (2020) offer insights over human-machine collaboration in craft and art-making. The central takeaway is to offer artists choices. *Devendorf* (2016) outlines relationship:

"Supporting a variety of practices in making is not a question of building

better tools or finding the 'killer' tool that subsumes all the others, but creating multiple pathways that honor the complexity, nonlinearity, and individuality of creative practices."

In the future, I hope to expand this study by examining the impact of GCDTs in low-resource art and craft domains. I refer to low-resource art and craft domains as craft disciplines with a small practitioner base and limited technological support. Because of the grammar-driven structure, GCDTs can potentially preserve craft knowledge. Particularly, I plan to adopt research methods similar to the ethnocomputing research process outlined in *Eglash et al.* (2006), which describes a pipeline for building Culturally Situated Design Tools (CSDTs). While craft domains analyzed in this dissertation do not have strong associations with specific culture and practitioner groups, I hope to explore the potential cultural impact of GCDTs in the future.

5.3 Conclusion

In this dissertation, I introduce Grammar-driven Craft Design Tools (GCDTs), which explicitly embed and utilize craft domain knowledge as their primary mechanisms and interfaces. Motivated by common challenges associated with computer-aided craft design tools, the GCDT framework bridge the gap between domain knowledge and design tool knowledge.

To provide an overview of how GCDTs can help artists find their creative expressions, I discuss three GCDTs developed in distinct craft domains. *InfiniteLayer* assists the design of multilayer sculpture by offering a set of grammar for describing

multilayer structures. *MarkMakerSquare* enables the creation of customized mark-making tools such as brushes and stamps using various materials. Lastly, *Thread-Plotter* combines hardware and software innovation to support plotter-made punch needle embroidery design and fabrication. I hope the GCDT framework will become a useful direction for building design-aid tools and supporting diverse artistic expressions in different craft domains.

BIBLIOGRAPHY

BIBLIOGRAPHY

Adamson, G. (2010), *The craft reader*, english ed. ed., Berg Publishers.

Aish, R., and R. Woodbury (2005), Multi-level interaction in parametric design, in *International symposium on smart graphics*, pp. 151–162, Springer.

Albaugh, L., S. E. Hudson, L. Yao, and L. Devendorf (2020), Investigating underdetermination through interactive computational handweaving, in *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, p. 1033–1046, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/3357236.3395538.

Ayres, J. (2001), *Monotype*, Watson-Guptill.

Baxter, W., and N. Govindaraju (2010), Simple data-driven modeling of brushes, in *Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games*, I3D '10, p. 135–142, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/1730804.1730826.

Berzowska, J., and M. Bromley (2007), Soft computation through conductive textiles, in *Proceedings of the International Foundation of Fashion Technology Institutes Conference*, pp. 12–15.

Birmingham, D. (2010), *Pop-up Design and Paper Mechanics: How to Make Folding Paper Sculpture*, Guild of Master Craftsman.

Bloch, J. (2006), How to design a good api and why it matters, in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, pp. 506–507.

Bostock, M., V. Ogievetsky, and J. Heer (2011), D^3 data-driven documents, *IEEE transactions on visualization and computer graphics*, *17*(12), 2301–2309.

Bronson, J., P. Rheingans, and M. Olano (2008), Semi-automatic stencil creation through error minimization, in *Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*, pp. 31–37.

Cannon, W. (1986), *How to Cast Small Metal and Rubber Parts*, McGraw-Hill Education.

Chamberlin, R., and M. Corbet (2017), *Beginner's Guide to Goldwork*, Search Press Classics, Search Press, Limited.

Chiang, Y. (1973), *Chinese calligraphy: An introduction to its aesthetic and technique*, vol. 60, Harvard University Press.

Chu, N. S. ., and Chiew-Lan Tai (2002), An efficient brush model for physically-based 3d painting, in *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.*, pp. 413–421, doi: 10.1109/PCCGA.2002.1167885.

Chung, J. J. Y., S. He, and E. Adar (2021), The intersection of users, roles, interactions, and technologies in creativity support tools, in *Proceedings of the 2021 DIS Conference on Designing Interactive Systems*.

Clary, C. (2013), Necro-diddlitis movement #1, `https://charlesclary.com/boxes/`, [Online; accessed 7-Feb-2021].

Cromwell, P. R. (2008), The distribution of knot types in celtic interlaced ornament, *Journal of Mathematics and the Arts*, *2*(2), 61–68.

Devendorf, L. K. (2016), Strange and unstable fabrication.

Dickinson, H. (1943), Besoms, brooms, brushes and pencils: The handicraft period, *Transactions of the Newcomen Society*, *24*(1), 99–108.

DiVerdi, S. (2013), A brush stroke synthesis toolbox, in *Image and Video-Based Artistic Stylisation*, pp. 23–44, Springer.

Eglash, R., A. Bennett, C. O'donnell, S. Jennings, and M. Cintorino (2006), Culturally situated design tools: Ethnocomputing from field site to classroom, *American anthropologist*, *108*(2), 347–362.

Eglash, R., L. Robert, A. Bennett, K. P. Robinson, M. Lachney, and W. Babbitt (2020), Automation for the artisanal economy: Enhancing the economic and environmental sustainability of crafting professions with human–machine collaboration, *AI & SOCIETY*, *35*(3), 595–609.

Fenghui Yao, and Guifeng Shao (2005), Painting brush control techniques in chinese painting robot, in *ROMAN 2005. IEEE International Workshop on Robot and Human Interactive Communication, 2005.*, pp. 462–467, doi: 10.1109/ROMAN.2005.1513822.

Fisher, T., and J. Botticello (2018), Machine-made lace, the spaces of skilled practices and the paradoxes of contemporary craft production, *cultural geographies*, *25*(1), 49–69.

Frankjær, R., and P. Dalsgaard (2018), Understanding craft-based inquiry in hci, in *Proceedings of the 2018 Designing Interactive Systems Conference*, pp. 473–484.

Frich, J., M. Mose Biskjaer, and P. Dalsgaard (2018), Twenty years of creativity research in human-computer interaction: Current state and future directions, in *Proceedings of the 2018 Designing Interactive Systems Conference*, pp. 1235–1257.

Frich, J., L. MacDonald Vermeulen, C. Remy, M. M. Biskjaer, and P. Dalsgaard (2019), Mapping the landscape of creativity support tools in hci, in *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, pp. 1–18.

Gamble, J. (2001), Modelling the invisible: The pedagogy of craft apprenticeship, *Studies in continuing education*, *23*(2), 185–200.

Gaver, W. (2012), What should we expect from research through design?, in *Proceedings of the SIGCHI conference on human factors in computing systems*, pp. 937–946.

Gray, R. M., and D. L. Neuhoff (1998), Quantization, *IEEE transactions on information theory*, *44*(6), 2325–2383.

Green, T. R. G., and M. Petre (1996), Usability analysis of visual programming environments: a 'cognitive dimensions' framework, *Journal of Visual Languages & Computing*, *7*(2), 131–174.

Griffiths, A. (1996), *Prints and Printmaking: An Introduction to the History and Techniques*, University of California Press.

Grishkoff, G. (2020), Mix media brushes & electrolux vacuum brushes, `https://www.glenngrishkoff.com/`, [Online; accessed 7-April-2021].

Guo, L., J. Peterson, W. Qureshi, A. Kalantar Mehrjerdi, M. Skrifvars, and L. Berglin (2011), Knitted wearable stretch sensor for breathing monitoring application, in *Ambience'11, Borås, Sweden, 2011.*

Hamdan, N. A.-h., S. Voelker, and J. Borchers (2018), Sketch&stitch: Interactive embroidery for e-textiles, in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–13.

Hanington, B., and B. Martin (2012), *Universal methods of design: 100 ways to research complex problems, develop innovative ideas, and design effective solutions*, Rockport Publishers.

He, S. (2015), Fruit study, watercolor painting. Artist's personal collection.

He, S., and E. Adar (2020), Plotting with thread: Fabricating delicate punch needle embroidery with x-y plotters, in *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, DIS '20, p. 1047–1057, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/3357236.3395540.

Heller, F., J. Thar, D. Lewandowski, M. Hartmann, P. Schoonbrood, S. Stönner, S. Voelker, and J. Borchers (2018), Cutcad-an open-source tool to design 3d objects in 2d, in *Proceedings of the 2018 Designing Interactive Systems Conference*, pp. 1135–1139.

Henning, M. (2009), Api design matters, *Communications of the ACM*, *52*(5), 46–56.

Higashi, T., and H. Kanai (2016), Instruction for paper-cutting: A system for learning experts' skills, in *Proceedings of the 2016 ACM International Conference on Interactive Surfaces and Spaces*, pp. 457–460.

Higashi, T., and H. Kanai (2019), Stylus knife: improving cutting skill in paper-cutting by implementing pressure control, in *Proceedings of the 34th ACM/SIGAPP Symposium on Applied Computing*, pp. 714–721.

Hudson, S. E. (2014), Printing teddy bears: a technique for 3d printing of soft interactive objects, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 459–468.

Iarussi, E., W. Li, and A. Bousseau (2015a), Wrapit: Computer-assisted crafting of wire wrapped jewelry, *ACM Trans. Graph.*, *34*(6), doi: 10.1145/2816795.2818118.

Iarussi, E., W. Li, and A. Bousseau (2015b), Wrapit: Computer-assisted crafting of wire wrapped jewelry, *ACM Trans. Graph.*, *34*(6), doi: 10.1145/2816795.2818118.

Ibbini, J., and S. Noyer (2021), Symbio vessels, `http://www.ibbini.com/vessels`, [Online; accessed 7-Feb-2021].

Igarashi, Y. (2011), Deco: A design editor for rhinestone decorations, *IEEE Computer Graphics and Applications*, *31*(5), 90–94.

Igarashi, Y. (2019), Bandweavy: interactive modeling for craft band design, *IEEE computer graphics and applications*, *39*(5), 96–103.

Igarashi, Y., and T. Igarashi (2010), Holly: A drawing editor for designing stencils, *IEEE Computer Graphics and Applications*, *30*(4), 8–14.

Igarashi, Y., T. Igarashi, and H. Suzuki (2008), Knitting a 3d model, in *Computer Graphics Forum*, vol. 27, pp. 1737–1743, Wiley Online Library.

Igarashi, Y., T. Igarashi, and H. Suzuki (2009), Interactive cover design considering physical constraints, in *Computer Graphics Forum*, vol. 28, pp. 1965–1973, Wiley Online Library.

Igarashi, Y., T. Igarashi, and J. Mitani (2012), Beady: interactive beadwork design and construction, *ACM Transactions on Graphics (TOG)*, *31*(4), 49.

Igarashi, Y., T. Hiyama, and K. Arakawa (2016a), An interactive system for original necklace design, in *ACM SIGGRAPH 2016 Posters*, pp. 1–2.

Igarashi, Y., T. Igarashi, and J. Mitani (2016b), Computational design of iris folding patterns, *Computational Visual Media*, *2*(4), 321–327.

Irvine, V., and F. Ruskey (2014), Developing a mathematical model for bobbin lace, *Journal of Mathematics and the Arts*, *8*(3-4), 95–110.

Jackson, D., J. Jackson, and R. Beer (2006), *Tibetan Thangka Painting: Methods & Materials*, Snow Lion Publications.

Jacobs, J., S. Gogia, R. Mundefinedch, and J. R. Brandt (2017), Supporting expressive procedural art creation through direct manipulation, in *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, p. 6330–6341, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/3025453.3025927.

Jacobs, J., J. Brandt, R. Mech, and M. Resnick (2018), Extending manual drawing practices with artist-centric programming tools, in *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–13.

Jacobs, J. J. M. (2013), Algorithmic craft: the synthesis of computational design, digital fabrication, and hand craft, Ph.D. thesis, Massachusetts Institute of Technology.

Jacobs, J. J. M. (2017), Dynamic drawing: Broadening practice and participation in procedural art, Ph.D. thesis, Massachusetts Institute of Technology.

Jain, A., C. Chen, T. Thormählen, D. Metaxas, and H.-P. Seidel (2015), Multi-layer stencil creation from images, *Comput. Graph.*, *48*(C), 11–22, doi: 10.1016/j.cag.2015.02.003.

Jeng-sheng Yeh, Ting-yu Lien, and Ming Ouhyoung (2002), On the effects of haptic display in brush and ink simulation for chinese painting and calligraphy, in *10th Pacific Conference on Computer Graphics and Applications, 2002. Proceedings.*, pp. 439–441, doi: 10.1109/PCCGA.2002.1167892.

Jung, S., Y.-S. Choi, and J.-S. Kim (2020), Stencil-based 3d facial relief creation from rgbd images for 3d printing, *ETRI Journal*, *42*(2), 272–281.

Kirby, R. (1950), Brush-making fibres, *Economic Botany*, *4*(3), 243–252.

Kono, T., and K. Watanabe (2017), Filum: A sewing technique to alter textile shapes, in *Adjunct Publication of the 30th Annual ACM Symposium on User Interface Software and Technology*, pp. 39–41.

Kryven, M., and E. Fourquet (2013), Generating knitting patterns from a sketch: a csp approach, in *Proceedings of the Symposium on Computational Aesthetics*, pp. 53–61, ACM.

Kubo, S. (2009), *Kami no japonisumu Kirié*, Tsuchiya Shoten.

Kumar, V. (2012), *101 design methods: A structured approach for driving innovation in your organization*, John Wiley & Sons.

Leaf, J., R. Wu, E. Schweickart, D. L. James, and S. Marschner (2018), Interactive design of periodic yarn-level cloth patterns, in *SIGGRAPH Asia 2018 Technical Papers*, p. 202, ACM.

Leaf, R. (1984), *Etching, engraving, and other intaglio printmaking techniques*, Dover Publications.

Levin, G., and T. Brain (2021), *Code as Creative Medium: A Handbook for Computational Art and Design*, MIT Press.

Levin, G., L. Huang, and T. Mustakos (2020), Pembroider, `https://github.com/CreativeInquiry/PEmbroider`, [Online; accessed 7-Feb-2021].

Li, J., J. Brandt, R. Mech, M. Agrawala, and J. Jacobs (2020), Supporting visual artists in programming through direct inspection and control of program execution, in *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*, pp. 1–12.

Li, Y., J. Yu, K.-l. Ma, and J. Shi (2007), 3d paper-cut modeling and animation, *Computer Animation and Virtual Worlds*, *18*(4-5), 395–403.

Likas, A., N. Vlassis, and J. J. Verbeek (2003), The global k-means clustering algorithm, *Pattern recognition*, *36*(2), 451–461.

Liu, E., L. Liu, J. Wang, Q. Jin, C. Yao, and F. Ying (2020), Int-papercut: An intelligent pattern generation with papercut style based on convolutional neural network, in *2020 15th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, pp. 59–67, IEEE.

Liu, L., Y. Chen, P. Wang, Y. Liu, C. Zhang, X. Li, C. Yao, and F. Ying (2018), Papercut: Digital fabrication and design for paper cutting, in *Extended Abstracts of the 2018 CHI Conference on Human Factors in Computing Systems*, pp. 1–6.

Liu, Y., J. Hays, Y.-Q. Xu, and H.-Y. Shum (2005), Digital papercutting, in *ACM SIGGRAPH 2005 Sketches*, pp. 99–es.

MacKenzie, G. (2010), *The watercolorist's essential notebook*, North Light Books.

Mahato, K. K., P. C. Kalita, and A. K. Das (2019), Design and development of affordable tool for metal handicraft, in *Research into Design for a Connected World*, pp. 357–367, Springer.

Malafouris, L. (2021), Mark making and human becoming, *Journal of Archaeological Method and Theory*, *28*(1), 95–119.

Markande, S. G., and E. A. Matsumoto (2020), Knotty knits are tangles on tori, *arXiv preprint arXiv:2002.01497*.

McCann, J., and D. Bryson (2009), *Smart clothes and wearable technology*, Elsevier.

McCarthy, L., C. Reas, and B. Fry (2015), *Getting started with P5. js: Making interactive graphics in JavaScript and processing*, Maker Media, Inc.

McCreight, T. (1994), *Practical Casting: A Studio Reference*, Jewelry Crafts, Brynmorgen Press.

McKinney, B. (2018), *ESTERBROOK a Dip Pen Legacy*, White Apple Multimedia.

Mei, P. (2019), Daily life, digital painting. Artist's personal collection.

Meng, M., M. Zhao, and S.-C. Zhu (2010), Artistic paper-cut of human portraits, in *Proceedings of the 18th ACM international conference on Multimedia*, pp. 931–934.

Mikkonen, J., and C. Fyhn (2020), *Storycoding - Programming Physical Artefacts for Research Through Design*, p. 441–455, Association for Computing Machinery, New York, NY, USA.

Mota, C. (2011), The rise of personal fabrication, in *Proceedings of the 8th ACM conference on Creativity and cognition*, pp. 279–288.

Mueller, S., N. Huebel, M. Waibel, and R. D'Andrea (2013), Robotic calligraphy — learning how to write single strokes of chinese and japanese characters, in *2013 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 1734–1739, doi: 10.1109/IROS.2013.6696583.

Mueller, S., B. Kruck, and P. Baudisch (2013), *LaserOrigami: Laser-Cutting 3D Objects*, p. 2585–2592, Association for Computing Machinery, New York, NY, USA.

Munzner, T. (2014), *Visualization analysis and design*, CRC press.

Nabil, S., J. Kučera, N. Karastathi, D. S. Kirk, and P. Wright (2019), Seamless seams: Crafting techniques for embedding fabrics with interactive actuation, in *Proceedings of the 2019 on Designing Interactive Systems Conference*, pp. 987–999.

Neddo, N. (2015), *The Organic Artist: Make Your Own Paint, Paper, Pigments, Prints and More from Nature*, Quarry Books.

Nitsche, M., and A. Weisling (2019), When is it not craft? materiality and mediation when craft and computing meet, in *Proceedings of the Thirteenth International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '19, p. 683–689, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/3294109.3295651.

Nitsche, M., A. Quitmeyer, K. Farina, S. Zwaan, and H. Y. Nam (2014), Teaching digital craft, in *CHI '14 Extended Abstracts on Human Factors in Computing Systems*, CHI EA '14, p. 719–730, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/2559206.2578872.

Noel, V. A. (2015), The bailey-derek grammar: recording the craft of wire-bending in the trinidad carnival, *Leonardo*, *48*(4), 357–365.

Oh, H., M. D. Gross, and M. Eisenberg (2015), Foldmecha: Design for linkage-based paper toys, in *Adjunct Proceedings of the 28th Annual ACM Symposium on User Interface Software & Technology*, UIST '15 Adjunct, p. 91–92, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/2815585.2815734.

Oxford, A. (2016), *Punch needle rug hooking: techniques and designs*, Schiffer Publishing.

Paczkowski, P., J. Dorsey, H. Rushmeier, and M. H. Kim (2018), Papercraft3d: paper-based 3d modeling and scene fabrication, *IEEE transactions on visualization and computer graphics*, *25*(4), 1717–1731.

Rael, R., and V. San Fratello (2018), *Printing architecture: Innovative recipes for 3D printing*, Chronicle Books.

Raji, R. K., X. Miao, S. Zhang, Y. Li, A. Wan, and A. Boakye (2019), Knitted piezoresistive strain sensor performance, impact of conductive area and profile design, *Journal of Industrial Textiles*, p. 1528083719837732.

Reas, C., and B. Fry (2007), *Processing: a programming handbook for visual designers and artists*, Mit Press.

Remy, C., L. MacDonald Vermeulen, J. Frich, M. M. Biskjaer, and P. Dalsgaard (2020), Evaluating creativity support tools in hci research, in *Proceedings of the 2020 ACM Designing Interactive Systems Conference*, pp. 457–476.

Rosner, D. K., and K. Ryokai (2009), Reflections on craft: Probing the creative process of everyday knitters, in *Proceedings of the Seventh ACM Conference on Creativity and Cognition*, C&C '09, p. 195–204, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/1640233.1640264.

Ross, B. (2017), *Bob Ross: The Joy of Painting*, Universe Publishing.

Ryan, R., and N. Avella (2011), *Paper Cutting Book: Contemporary Artists, Timeless Craft*, Chronicle Books.

Satyanarayan, A., D. Moritz, K. Wongsuphasawat, and J. Heer (2016), Vega-lite: A grammar of interactive graphics, *IEEE transactions on visualization and computer graphics*, *23*(1), 341–350.

Scalera, L., S. Seriani, A. Gasparetto, and P. Gallina (2019), Watercolour robotic painting: a novel automatic system for artistic rendering, *Journal of Intelligent & Robotic Systems*, *95*(3-4), 871–886.

Schama, G. (2019), Autophagocytosis, `https://www.gabrielschama.com/2018-2019/2019/4/22/autophagocytosis`, [Online; accessed 7-Feb-2021].

Schoning, J., Y. Rogers, and A. Kruger (2012), Digitally enhanced food, *IEEE Pervasive Computing*, *11*(3), 4–6.

Shiner, L. (2012), "blurred boundaries"? rethinking the concept of craft and its relation to art and design, *Philosophy Compass*, *7*(4), 230–244.

Skouras, M., S. Coros, E. Grinspun, and B. Thomaszewski (2015), Interactive surface design with interlocking elements, *ACM Transactions on Graphics (TOG)*, *34*(6), 224.

Standley, E. (2020), Llull, `http://www.eric-standley.com/#/llull/`, [Online; accessed 7-Feb-2021].

Stewart, M. (2009), *Punchneedle The Complete Guide*, Penguin.

Stewart, M. (2013), *Easy, elegant punchneedle*, Stackpole Books.

Stylos, J., and B. Myers (2007), Mapping the space of api design decisions, in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2007)*, pp. 50–60, IEEE.

Tanenbaum, J. G., A. M. Williams, A. Desjardins, and K. Tanenbaum (2013), Democratizing technology: pleasure, utility and expressiveness in diy and maker practice, in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2603–2612.

Tomsky, M. (2019), Fox and hare earrings - woodland collection, `https://www.martintomsky.com/woodland-collection/ujgd76ik3gsq49t00kr6zw0sqo4xrz`, [Online; accessed 7-Feb-2021].

Tomsky, M. (2020), Haunted house, https://www.martintomsky.com/fantastical/p40bqt4y7mjdcfo8ujuwps2ab0rwnk, [Online; accessed 7-Feb-2021].

Torres, C., W. Li, and E. Paulos (2016), Proxyprint: Supporting crafting practice through physical computational proxies, in *Proceedings of the 2016 ACM Conference on Designing Interactive Systems*, pp. 158–169, ACM.

Tsolis, A., W. G. Whittow, A. A. Alexandridis, and J. Vardaxoglou (2014), Embroidery and related manufacturing techniques for wearable antennas: challenges and opportunities, *Electronics*, *3*(2), 314–338.

Tung, F.-W. (2012), Weaving with rush: Exploring craft-design collaborations in revitalizing a local craft, *International Journal of Design*, *6*(3).

Vantours, M. (2019), Spirales, https://maudvantours.com/portfolio-item/spirales/, [Online; accessed 7-Feb-2021].

Vantours, M. (2020), Hetch x formica, https://maudvantours.com/portfolio-item/hetch-x-formica/, [Online; accessed 7-Feb-2021].

Vernon, V. (2013), *Implementing domain-driven design*, Addison-Wesley.

Wax, C. (1996), *The mezzotint: history and technique*, H.N. Abrams.

Wilkinson, L. (2012), The grammar of graphics, in *Handbook of Computational Statistics*, pp. 375–414, Springer.

Willis, S. (2018), The maker revolution, *Computer*, *51*(3), 62–65.

Winters, E. (2014), *Mastering copperplate calligraphy: a step-by-step manual*, Dover Publications.

Wu, K., X. Gao, Z. Ferguson, D. Panozzo, and C. Yuksel (2018), Stitch meshing, *ACM Transactions on Graphics (TOG)*, *37*(4), 130.

Wu, K., H. Swan, and C. Yuksel (2019), Knittable stitch meshes, *ACM Transactions on Graphics (TOG)*, *38*(1), 1–13.

Xu, J., C. S. Kaplan, and X. Mi (2007), Computer-generated papercutting, in *15th Pacific Conference on Computer Graphics and Applications (PG'07)*, pp. 343–350, IEEE.

Yang, J., S. He, and L. Lu (2019), Binary image carving for 3d printing, *Computer-Aided Design*, *114*, 191–201.

Zhang, M., Y. Igarashi, Y. Kanamori, and J. Mitani (2015), Designing mini block artwork from colored mesh, in *International Symposium on Smart Graphics*, pp. 3–15, Springer.

Zheng, C., and M. Nitsche (2017), Combining practices in craft and design, in *Proceedings of the Eleventh International Conference on Tangible, Embedded, and Embodied Interaction*, TEI '17, p. 331–340, Association for Computing Machinery, New York, NY, USA, doi: 10.1145/3024969.3024973.