Foundations for 3D Machine Knitting

Vidya Narayanan

CMU-CS-21-140
September 2021

Computer Science Department
School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

*Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.*

This research was sponsored by Shima Seiki Mfg LTD and the National Science Foundation under award numbers IIS1566244 and IIS1955444. The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: Computer-aided Manufacturing, Computer-aided Design, Machine Knitting, 3D Printing, 3D Modeling, Graphics systems and interfaces

Abstract

Industrial knitting machines are programmable systems that can fabricate complex soft objects. However, this traditional manufacturing system has largely been overlooked as a technology for custom and rapid fabrication. For machine knitting to be widely adopted for custom fabrication, end users must be allowed to work in an intuitive design space instead of low-level machine operations.

In this thesis, I lay out foundational tools for machine knitting that allow users to think about *what* they want to create in terms of 3D shapes instead of *how* the machine constructs by showing that programming for machine knitting can be organized to decouple high-level design challenges from low-level machine input decisions.

Such an organization allows exploration of various aspects of machine-knitting independently: pattern design, layout planning, and machine-code generation. I classify the space of shapes that can be constructed with industrial knitting machines. I present a new data-structure to represent 3D shapes as knitting programs. I describe geometric algorithms to create patterns from 3D models and an editing framework to modify patterns in 3D. For fabrication, I describe a scheduling algorithm to translate patterns into low-level machine code. Together, these tools and techniques allow designers and end-users to treat 3D machine knitting as an accessible 3D-printing-like soft fabrication system.

Acknowledgments

This thesis would not have been possible without my advisor, Jim McCann. I began working with Jim at Disney Research Pittsburgh when the Textiles Lab was just starting out. From then to now it has been a wonderful six years. I have learnt so much from Jim and his eclectic interests and I hope to continue to do so. Jim has always encouraged me to think deeply, write (and code) simply, not worry too much about conference deadlines, and most importantly, have fun working. I could not have asked for a better advisor and I am proud to be his student. Jessica Hodgins has been a really positive influence on my work both at Disney Research and CMU. I always appreciate her direct and precise suggestions. I would like to thank my entire committee: Jessica Hodgins, Keenan Crane, Adriana Schulz, and Jim McCann, for their support. Their insights and feedback has helped shape this thesis immensely. I am very glad that I got to learn and work with Stelian Coros during my initial years as a graduate student. He brought an infectious positive energy to projects that made working with him so easy. I would also like to thank my masters advisor, Vijay Natarajan, for introducing me to research and encouraging me to pursue a Ph.D; and Kayvon Fatahalian for introducing me to folks at Disney Research which played a huge role in my decision to move to Pittsburgh.

I have had the privilege to work with many, many collaborators over the past few years and this work would not have been possible without them: Lea Albaugh, Kui Wu, Cem Yuksel, David Breen, Jianzhe Gu, Lining Yao, Jenny Lin, Yuka Ikarashi, Gilbert Bernstein, Jonathan Regan-Kelley, Michelle Guo, April Grow, Jennifer Mankoff, Wojciech Matusik. Thank you, all! Especially Lea, for teaching me so much about illustrations and always being there around deadlines. I spent three wonderful months interning at Adobe Research. Thank you Michal Lukac, Danny Kaufmann, Amanda Ghassaei and the many folks at Adobe Research for working with me on the delightful box folding project. Thank you, Amanda, for being always available to chat about research and life. Working with Shima Seiki, Japan, has been an incredible experience. I am grateful to Mr. Terai and the entire team at Shima Seiki, for supporting my research, hosting us at Wakayama, and for making a really great knitting machine! Thank you, Deb Cavlovich, for always being available and knowing everything; Catherine Copetas, for helping with all things related to submitting a thesis; Brian Hutchison, for taking care of all the administrative work and always being patient and kind.

Being in the Graphics Lab at CMU has been thoroughly enjoyable. Lunch and ramblings with Chris Yu, Jenny Lin, Evan Shimizu, Arjun Teh, Mark Gillespie, Rohan Sawhney, Ruta Desai, Yanzhe Yang, Nick Sharp, Jim Bern, and Fait Poms were always fun. It has always been exciting to discuss and learn from Yannis Gkioulekas, Srinivas Narasimhan, Keenan Crane, Matthew O'Toole, Nancy Pollard, Jessica Hodgins and Kayvon Fatahalian. The Textiles Lab has been home, thank you, Lea Albaugh, Nur Yildirm, Jenny Lin, Ella Moore, Evan Shimizu, Gabrielle Ohlson, Michelle Guo, Yixin He, Catherine Yu, Anne He, Ticha Sethapakdi, April Grow, Chenxi Liu and of course, Jim, for making it what it is. I would like to thank my friends: Purvi, Mitul, Ana, Saurabh, and Lipika, for keeping me sane, especially over the last two extraordinary years (and for entertaining all the quizzes). I would also like to thank the doctors of UPMC for fixing me, now and then.

My family has always been there for me even through decisions they didn't quite understand. Amma, Papaji, Ammamma, and Patti, thank you for everything. I have always looked up to my sister, Srikala, the first and real doctor in the family. Sri and Ashish have always been there for me. My nephew and niece, Siddharth and Saina, bring so much joy and make everything meaningful. I'm glad that we could spend so much time together in Pittsburgh! My in-laws have been supportive of everything I've done, thank you for never forgetting to send boxes of delicious homemade snacks. Finally, I'd like to thank my husband, Ravi

Teja, who has been a true partner in every sense of the word. I would not have taken this path if it was not for him, and I am so happy that I did.

Contents

1 Introduction 9

1.1 Contributions and Organization 12

2 An Introduction to Machine Knitting 15

2.1 Knitting 15

2.2 Machine Knitting 15

2.3 Emulating a multi-layer knitting machine 18

2.4 Knitout : An assembly language for machine knitting 19

2.5 Miscellaneous 21

3 Background 23

3.1 Knitting 23

3.2 Clothing Design 28

3.3 Other Soft Fabrication Systems 28

3.4 Simulation and Rendering 29

4 What Can Be Machine Knit ? 31

4.1 Design Space of a (Multi-Layer) Machine 32

4.2 Physical Constraints 36

5 Representing and Editing Machine Knitting Patterns 39

5.1 The augmented stitch mesh 40

5.2 Editing an Augmented Stitch Mesh 44

6 Constructing Machine Knitting Patterns from 3D meshes 53

6.1 General Patterns 54

6.2 Transfer-Free Patterns 66

7 Turning Knitting Patterns into Machine Code 75

7.1 User-in-the-loop embedding for general structures 79

7.2 Explicit embedding of tubular layouts for two-layer machines 87

7.3 Code Generation 92

8 Knit Results 95

8.1 General two-bed patterns (Incremental approach) 95

8.2 General two-bed patterns (hierarchical approach with user editing) 98

8.3 Transfer-free patterns 103

8.4 Multi-layer patterns with user editing 104

9 Discussion 107

1

Introduction

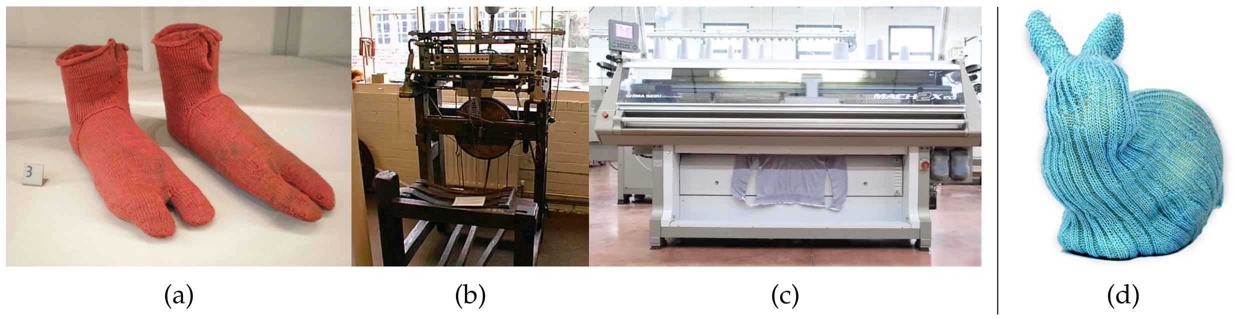


Figure 1.1: A few snapshots in the timeline of industrial knitting advances: (a) Socks created with nålebinding (a precursor to knitting) have been discovered and dated from around 300 A.D. (b) William Lee's stocking frame launched industrial machine-knitting in 1589 (c) Shima Seiki introduced seamless machine knitting in 1995. A Shima Seiki MACH 2X seamless machine shows a fully shaped 3D sweater being knit. (d) This thesis: A custom "3D Knit" Stanford Bunny designed from a 3D mesh using tools described in this thesis and fabricated on an industrial knitting machine.

Industrial knitting machines – which produce the sweaters and upholstery we use everyday – are a general, programmable, soft-fabrication system. These industrial manufacturing systems have a rich, long history and are reliably used for mass production today (Figure 1.1). However, they are often not thought of as candidates for custom fabrication. A key reason for this situation is that designing patterns for such systems requires expertise in knitting as a fabrication technique and knowledge of hardware-specific instructions. In this thesis, I describe how industrial machine knitting can be made more accessible by viewing the machine as a programmable system and building layers of abstractions that separate high-level challenges (e.g., designing a bunny with different textures) from low-level machine operations that execute the pattern (e.g., add loops on needles 1 to 10).

Thesis: Programming for machine knitting can be organized to decouple high-level design challenges from low-level machine input decisions.

Today, industrial machine knitting caters to a wide range of applications including use in industrial composite reinforcements (Ramakrishna [1997], Rudd et al. [1990]); architecture (Popescu [2019]);

medicine and health-care (Zhang and Ma [2018]); robotics (Maziz et al. [2017]); and, of course, upholstery, accessories and garments (Turney [2009]). For most of these industries, custom one-off fabrication and personalization would improve impact (Adidas [2019], Cross and Podhajny [2017], Ministry of Supply, Unmade). Despite this, these manufacturing systems have been largely overlooked as avenues for rapid and custom fabrication and often rely on expert-designed mass-produced solutions.

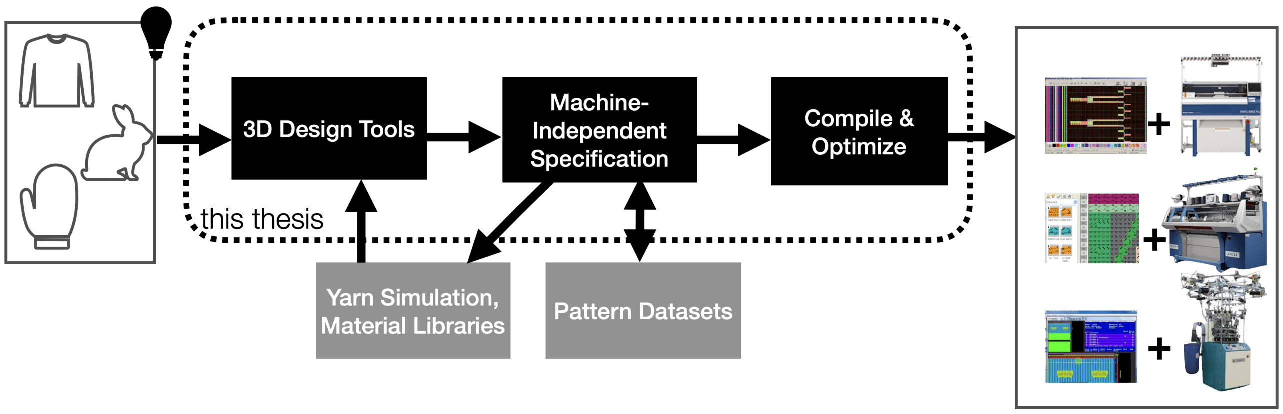

3D Design Tools
Machine-Independent Specification
Compile & Optimize
this thesis
Yarn Simulation, Material Libraries
Pattern Datasets

Figure 1.2: An effective ecosystem for designing knit structures: beginning from an idea, high-level design tools assist designers at various levels of expertise to generate and edit a machine-independent representation. Yarn simulation, material libraries and pattern datasets are incorporated for design assistance, visualizations and previews. Finally, low-level representations are automatically compiled and optimized into knitting code for one or more available machines.

We observed the following at two US-based knitting manufacturers: To (mass) produce a knit accessory like a glove or sock, first an appropriate manufacturing machine is picked. Then, designs are created using expert-guided templates which are usually machine-specific. These patterns are tuned based on design and material parameters often requiring multiple iterations of fabrication. Finally once the pattern specification matches all requirements, many thousands of artifacts can be knit, amortizing the design cost.

An alternate manufacturing pipeline might look like Figure 1.2: the end-user starts with a 3D model (acquired e.g.,by scanning, downloading from the internet, or having an expert generate a CAD design) and picks a few high-level design parameter values (such as yarn type and knitting gauge). Using this high-level input, a computational design tool generates a custom pattern that is compatible with machine-knitting. Such a tool would also allow the designer to quickly edit and visualize this pattern if necessary. The user then finds an appropriate (potentially local) manufacturing unit to produce and ship the product after fabrication (similar to what Shapeways[1] has established for 3D Printing)). Manufacturers can optimize and convert the user-specified design representation to the appropriate hardware specification based on machine availability.

[1] https://www.shapeways.com/

One significant impediment to the existence of such an ecosystem is that the current design tools are tied closely to machine operations. To design a complex 3D shape with current knitting CAD systems, the designer must necessarily have a clear picture of how the machine will execute the shape while designing. For example, Figure 1.3 shows a knitting pattern for a glove designed in KnitPaint (Shima Seiki [2019]). The designer creates a flat, image-based program where each pixel encodes one or more machine operations. Stitch creation and loop movements are encoded by the color of the pixel and its location in the 2D image. For standard designs, 3D views may be available to visualize the pattern.

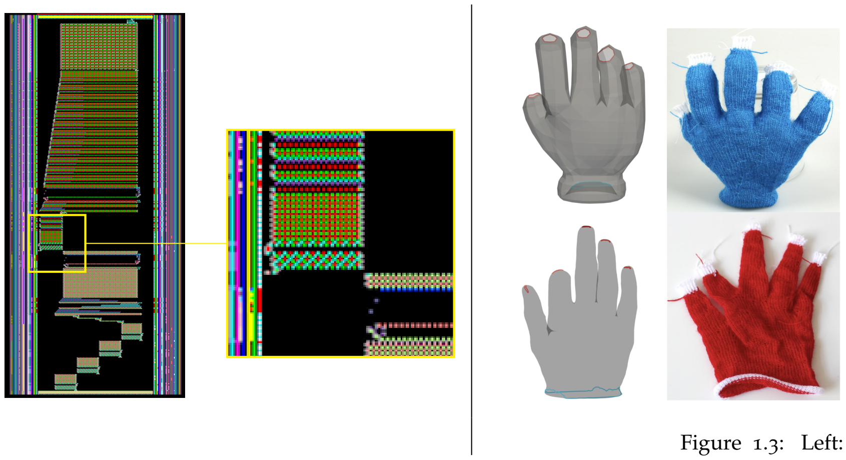


Figure 1.3: Left: KnitPaint (A part of the SDS design suite from Shima Seiki) requires programming patterns using a image-based language. For this glove pattern (generated from a template), each colored pixel encodes one or more hardware operations. Right: Using 3D shapes and a few high-level design hints as input to directly generate glove patterns from a mesh of a cartoon glove (blue) and that of a scanned hand (red).

In contrast, we envision a system that allows designers to work in the natural 3D space of the output shape instead of the working space of the machine. This requires introducing appropriate layers of abstractions that allow designers to interact with the knitting system more intuitively.

Other fabrication systems further reinforce the benefits of building levels of abstractions when interacting with the system. At a high level, 3D printers allow casual users to supply a 3D model as an input. Intermediate computational tools can identify where support structures may be needed and automatically place them, generate in-fill structures and optimize the input in useful ways. Next, appropriate slicing algorithms are used to generate machine compatible G-Code (Cura [2018], Ranellucci [2013], Schmidt and Singh [2010]). CNC-Milling softwares similarly allow end-users to load up a 3D CAD model designed in a solid modeling software (AutoCAD, Fusion360, SolidWorks) and pick a series of tools to machine the surface with. Tool paths are automatically generated and simulations of the tool in action are presented (MasterCam). Users with varying levels of expertise may edit the execution at various stages with the appropriate tools.

More generally, even in other areas of computing, high-level and domain-specific programming languages coupled with effective compiler techniques that abstract low-level languages have been key in improving productivity and creating complex systems (Hu et al. [2019], Kjolstad et al. [2016], Nandi et al. [2017], Ragan-Kelley et al. [2012]).

For 3D knitting to be successful as a general soft-fabrication system, end-users must be allowed to think in a rich and intuitive but feasible design space i.e., *what* they want to make instead of *how* a machine makes it; with effective CAD tools that navigate the gap between the two.

1.1 Contributions and Organization

This thesis is broadly organized as follows. I first introduce machine knitting as a programmable system (Chapter 2) and discuss related work in the area (Chapter 3). I then describe the main contributions of this thesis:

Ch. 4 **What Can Be Machine Knit ?** I model the constraints of the industrial knitting machine as a generalized *multi layer* knitting system. From these scheduling constraints, I formalize the space of shapes that can be machine knit in terms of its topology and discuss discrete constraints that come into play for fabrication.

Parts of this chapter is based on work previously presented in Narayanan et al. [2018]

This classification identifies the high-level design space of machine-knittable shapes. This is a key step in ensuring that design systems can be constrained to create and support valid high-level designs without needing to specify stitch-level details.

Ch. 5 **Representing and Editing Machine Knitting Patterns** Next, I describe a constrained but intuitive geometric data structure – the Augmented Stitch Mesh – that allows end-users to design in the 3D output space of the shape instead of the traditional machine space.

Parts of this chapter has been presented in Narayanan et al. [2018, 2019]

This data structure can support constrained editing for both shape and texture. I introduce an interactive design system to support *creating* and *editing* arbitrary knitting operations on surfaces.

Ch. 6 **Constructing Machine Knitting Patterns from 3D meshes**

For machine knitting to be accessible, constructing augmented stitch meshes must be easy. I describe two styles of geometric algorithms to translate oriented, manifold 3D surfaces into augmented stitch meshes. The first focuses on surfaces with disk-like topology that can be constructed with simple single-bed consumer machines i.e., without any transfer operations. The second produces general 3D surfaces as a combination of tubes that can be fabricated with industrial two-bed knitting machines with transfer capabilities.

Ch. 7 **Turning Knitting Patterns into Machine Code** For fabrication on the machine, the augmented stitch mesh needs to be scheduled: stitches need to be assigned to machine needles and any operations needed to move loops around need to be identified.

Parts of this chapter is based on McCann et al. [2016], Narayanan et al. [2018]

For patterns that are a combination of tube-like structures, I describe a scheduling algorithm that enumerates all possible cyclic layouts for loop placement on a two-bed machine. Using these layouts as input to the augmented stitch mesh representation, machine level knitting code for stitch construction is generated. Loops may need to be moved to match positions between layouts. This is performed by a transfer-planning algorithm that introduces low-level loop-movement code that is interleaved with the stitch generation code for generating a complete machine program. For constructing general patterns, I present a user-in-the-loop scheduling setup, where user-generated annotations are used to guide scheduling and transfer planning.

In chapter 8, I discuss a series of knit results fabricated using the techniques presented, and, discuss directions to expand on these ideas in chapter 9.

2
An Introduction to Machine Knitting

I now briefly introduce knitting as a fabrication technique, discuss how to model a knitting machine, and express it as a programmable system with a small instruction set.

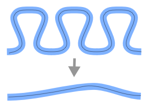


Figure 2.1: In knitting, yarn is shaped into loops to form stitches. Isolated loops can be unraveled by tugging the yarn.

2.1 *Knitting*

Knitting involves forming (two-dimensional) fabric by manipulating (one-dimensional) yarn. Yarn is turned into *loops* which are pulled through other loops to create a stable 2D surface.

A single piece of yarn can be turned into a piece of fabric by creating *courses* (rows) of loops, each course stabilized by the ones before and after it as shown in Figures 2 and 2. *Wales* (columns) arise from following each loop's dependencies. Figure 2 illustrate some of these common knitting terms. Although the term stitch and loop are colloquially interchangeable, we use the term *stitch* to describe one or more loops with their dependencies.

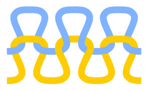


Figure 2.2: Loops that have been pulled through other loops form stable stitches. Here, the yellow loops rest securely on the blue loops that have been pulled through them.

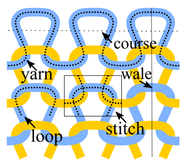


Figure 2.3: Common knitting terms: A row of loops that are made in sequence is called a course and column of loop dependencies forms a wale. One or more loops with all their dependencies constitute a stitch.

The "loop-through-loop" stabilizing structure is the basic building block of a knit fabric. This structure can be created by hand (usually using long needles or peg-looms to hold unstable loops), or by using various mechanized systems.

2.2 *Machine Knitting*

Any (weft) knitting machine[1] has to manipulate yarn and turn them into stitches. Individual hook-shaped *needles* (illustrated in figure 2.4) are used for creating and holding loops. Each needle can make a loop by grabbing yarn that appears before it and *tucking* it on to its hook. The needle can also *pull* the loop through all the other loops it holds to produce a *knit* stitch.

[1] Warp knitting machines also create fabric from interconnected loops. They use a fixed number of wales (separate yarns) and are used more commonly to produce cloth of a constant width. In this thesis, I use the term machine knitting to refer to weft knitting.

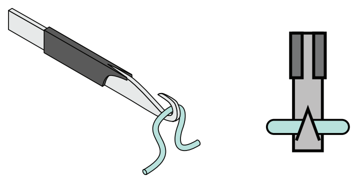


Figure 2.4: Knitting machines use hook-shaped needles to hold loops.

These needles can be positioned in different ways to construct different types of knitting machines (Figure 2.5).

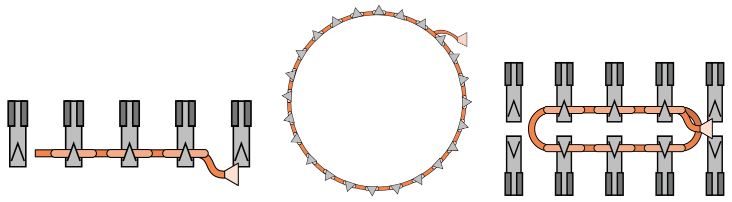


Figure 2.5: Different needle layouts for different styles of machines. (Left) A single linear layout for flat knitting, (Middle) Circular layout for seamless tubular knitting, (Right) Two bed layout for whole garment knitting

In the simplest layout, needles are arranged linearly one after the other (often called a *bed*). Such a machine is called a *flat* knitting machine since it can produce sheets of flat fabric.

Knitting on such a machine takes place like so (see Figure 2.6):

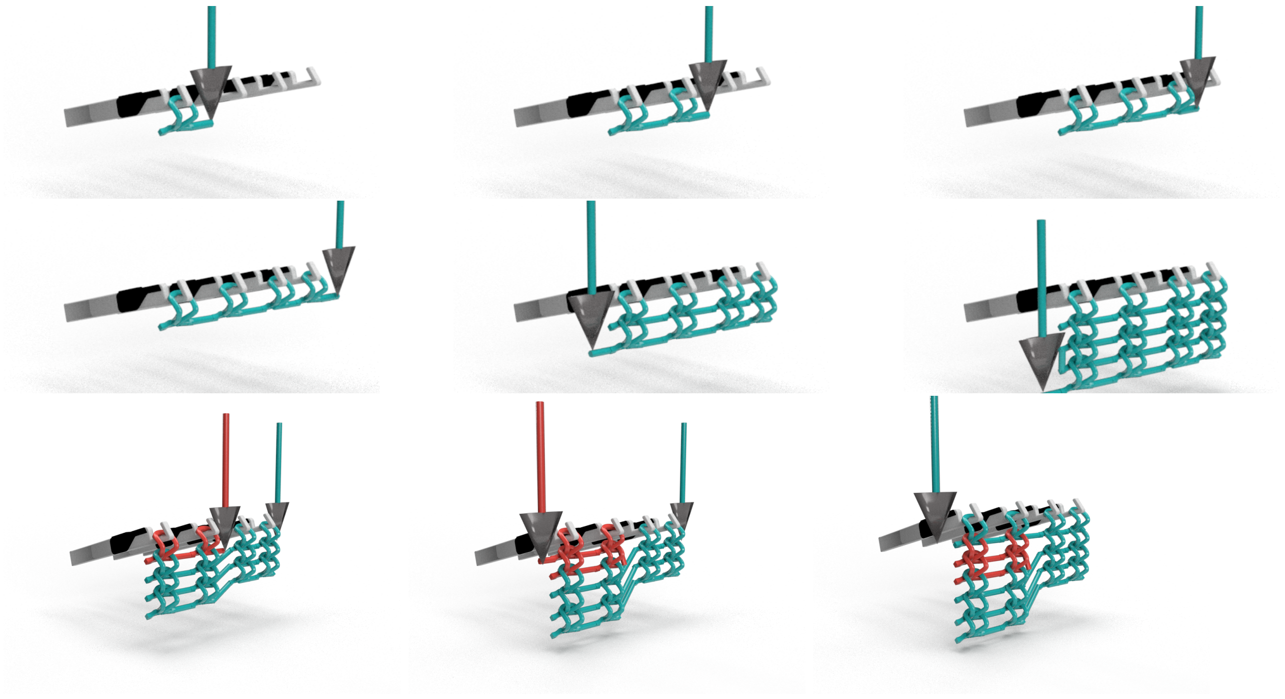


Figure 2.6: Fabricating a flat sheet on a single bed knitting machine – the yarn carrier moves across the needles, that form and hold knit stitches. Some columns can be lengthened more than others by making shorter trips with the yarn carrier as shown here with the red yarn.

Yarn enters the machine from a cone, passing through a tensioning device and a *yarn carrier* shown as a triangle in the figure above. Yarn carriers move laterally across the needle bed(s), positioning new yarn where it is needed. To actuate the needles, a *carriage* passes over the needle bed with cam-plate mechanisms that can lower and raise the needles.

In practice, knitting machines incorporate *many* other important components such as sinkers and loop-pressers to secure and form loops, fabric pressers to push down on the fabric being formed between the beds, various forms of take-down and tensioning devices and sensors that identify knots or slack in the yarn. Spencer [2001] provides a detailed introduction to the various components of machine knitting.

A single flat bed machine can make rectangular sheets of fabric by knitting over a sequence of needles in one direction, then reversing direction knitting over the same needles in reverse order (repeating this to the desired length). Additionally, it can make complex 3D surfaces with a disk topology by creating *short rows*. A *short row* (also called *partial knitting* or *flechage*) refers to knitting over a small portion of the currently active needles holding loops while leaving the rest of the loops on the needle-bed thereby lengthening a few columns more than others and adding curvature (see the red yarn in Figure 2.6).

To produce single tubular structures, the needles can be arranged in a circle. Such a machine is similar to a linear knitting machine, but can now hold a seamless tubular structure. These are commonly used for manufacturing socks, heels are created using short-rows.

Although both flat and circular machines are highly prevalent in the industry, the state of the art industrial hardware uses two linear beds arranged facing each other in an inverted-V shape and called *V-bed* or two-bed knitting machines. The distance between needles and between the front and back bed are nearly equal, such that length of the yarn between loops placed directly across the bed and along two adjacent needles on the same bed are approximately the same. Each needle can not only add loops and create knit stitches, but can also *transfer* (move) loops held on its hook to the needle across it on the opposite bed. The back bed can also be translated or *racked* relative to the front bed. Combining these two abilities, loops can be moved around in a two-bed machine. By moving loops around, the width of a fabric can be increased or decreased. To increase the fabric width, a gap is introduced by moving a few loops one (or more) needles over and a new loop is added to the empty location in the next row of knitting. A swatch with an increase is shown in figure 2.7

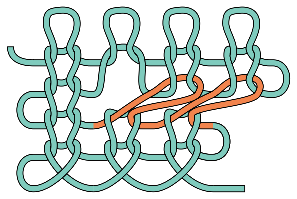


Figure 2.7: Increasing the width of a fabric by moving loops to create a gap

To decrease the fabric width, two (or more) loops are moved to the same needle, the next knit operation through that needle pulls a loop through all the overlapped loops. Figure 2.8 shows a three-stitch wide fabric narrowed into a two-stitch wide fabric with decreases.

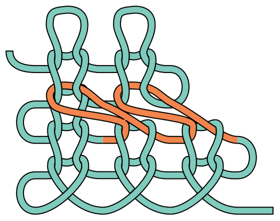


Figure 2.8: Decreasing the width of a fabric by moving loops and overlapping them

Multiple tubular structures can be created on these machines by using both beds to hold loops – squishing a cyclic layout onto the two beds. Because a 3D tubular piece is constructed without the need to seam pieces of fabric together, these are also called *seamless* machines.

It might (correctly) seem that shaping a cycle when both the beds have stitches is difficult without causing a tangling of yarn (see figure 2.9. This can be resolved by a four-bed (also called an X-bed) machine which adds an extra pair of beds to deal with this situation – two parallel beds are used for knitting and two additional beds are used for temporary storage.

In order to avoid this complexity in representation (and to mimic this behavior on simpler form of the machine) we often resort to a machine-knitting trick: *half-gauge* knitting. In such a setup, we assume that the front bed consists of even-numbered knitting needles and the back bed consists of odd-numbered knitting needles (colored dark in the illustration 2.10). The odd-numbered front bed needles (aligned with the back-bed knitting needles and vice-versa) act as temporary holding positions (colored light in the illustration 2.10). Loops can be moved from the knitting needle to the associated holding position on the opposite bed.

This idea of using alternate needles can be further generalized to turn any two bed machine into a general N-layer machine.

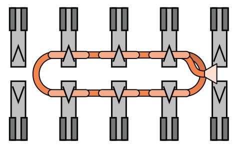


Figure 2.9: A tube on a double bed machine. This tube cannot be shifted or translated by one needle because transfers will overlap loops that cannot be later separated.

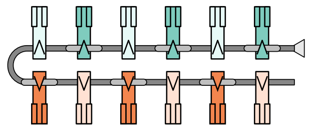
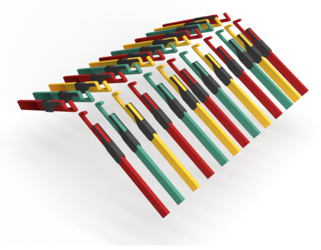


Figure 2.10: Using alternate needles for holding loops and empty ones for temporary storage during transfers.

2.3 Emulating a multi-layer knitting machine

To emulate an N-layer machine on a two-bed knitting machine, needles of *both* beds are partitioned into N interleaved sets – the i^{th} needle belongs to the i^{th} layer. In Figure 2.11 these partitions are color-coded.



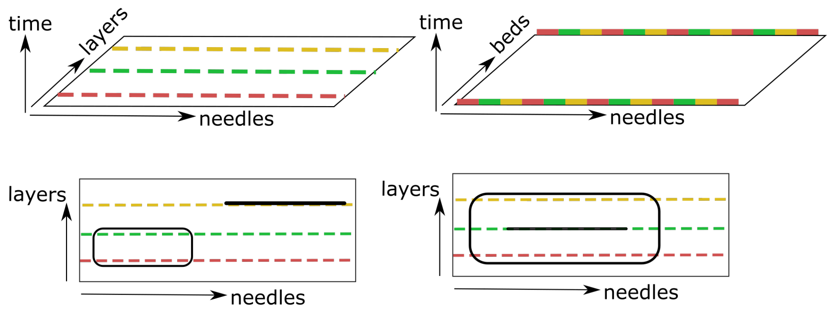


Figure 2.11: A three layer machine can be emulated by interleaving three different needle sets. By maintaining the relative ordering of the layers, a tube and a sheet can be held at the same time with enough degrees of freedom to shape them.

With such a setup, two *layers* are required to hold a tubular cycle and one layer can hold a sheet[2]. The needles on the back bed of a front layer and the needles on the front bed of a back layer act as temporary storage. The half-gauge machine described above is an instance of a two-layer knitting machine and can hold tubes that can be shaped with increases and decreases. A three layer machine similarly can be used to hold 1 tube and 1 sheet or 3 sheets in parallel as shown in the illustration above.

[2] A fixed tube can be held on a single layer since the layer includes needles from both beds, but such a tube cannot be moved and shaped.

An important constraint in this setup is that a front-to-back ordering of layers must be maintained throughout the knitting process to avoid *tangling* where loops from one layer get captured between loops of other layers irreversibly pinching layers together. Given an ordering on the layers, a front-bed operation can be performed on the layer only after all layers with a higher index are moved (using transfers) to the back bed. A back-bed operation can be performed on a layer only after all layers with lower index are moved to the front bed. This flipping of layers ensures that the yarn-carrier is placed in a way that does not cause it capture loops from a different layer while knitting on a layer. In chapter 5 and 7, I will show that these constraints can be systematically expressed within a data-structure for representing knitting programs or patterns.

In order to talk about these knitting operations clearly and program the machine, we first need a language that can describe all the operations a knitting machine can perform.

2.4 *Knitout : An assembly language for machine knitting*

I'll now explicitly describe an "assembly language" called *knitout* to operate a general industrial knitting machine. *Knitout* is a machine independent format for supporting machine knitting that we developed at Disney Research Pittsburgh and the Carnegie Mellon Textiles Lab (McCann [2017]).

We use the following convention to identify needles (on an unbounded needle-bed):

$$\forall i \in \mathbb{Z} : \begin{cases} \texttt{b}i\text{: back bed needle} \\ \texttt{f}i\text{: front bed needle} \end{cases}$$

Needles are indexed left-to-right along the bed, and are aligned across the bed i.e., $\texttt{f}-2$ is aligned with $\texttt{b}-2$, which is to the left of $\texttt{b}-1$. Needle $\texttt{f}2$ is considered adjacent to needles $\texttt{f}1$, $\texttt{f}3$ and $\texttt{b}2$ because they have the same spacing between them.

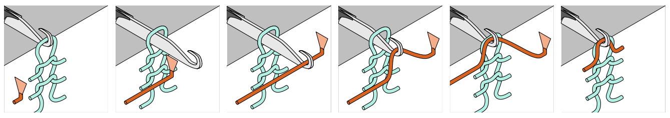
We use this ordering of needles to describe the *direction* in which the yarn (carriers) move – a *positive* $(+)$ direction if the needle indices increase and a *negative* $(-)$ direction if the needle indices decrease. Multiple yarn carriers may move in tandem and we refer to the active yarn carriers at any instant as the yarn *carrier set*.

We now describe the basic needle and state operations on the machine.

Knitout Syntax

```
tuck D N CS
```

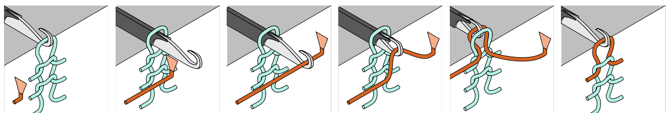
Tuck. The tuck operation adds a new loop of yarn in front of the loops already held on a needle. Mechanically, the needle reaches forward, the yarn carrier moves to the right over the needle, and the needle retracts, now holding a new loop:



In the knitout specification, the tuck operation is defined by specifying the direction **`D`** of yarn-carriers , needle-bed position **`N`** and yarn-carrier set **`CS`** participating in the operation.

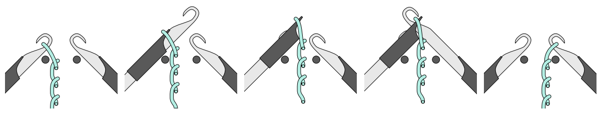
`knit D N CS`

Knit. Knitting a needle pulls a new loop of yarn through the *all of the loops* currently held by that needle. Mechanically, the needle reaches forward, the yarn carrier moves over it, and the needle retracts, using a secondary mechanical action to lift the loops that it was holding up and over the new loop and off of its tip.



`xfer N1 N2`

Transfer. The transfer operation moves all the loops on a needle to the needle across from it. That is, it moves loops from the front bed to the back bed or visa versa.

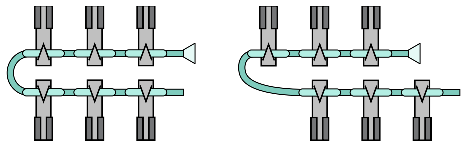


In the knitout specification, the transfer operation is defined by specifying the needle-bed position N_1 of the source loop(s) and the needle-bed position N_2 of the target loop(s).

This restriction of only moving between aligned needles may seem severe, but machines can *rack* (laterally move) the beds to change which needles are aligned. By convention, the racking amount is the offset of the back bed with respect to the front.

`rack:amount ;`

Rack. The *rack* operation changes the machine state by translating the back bed by the specified amount changing the alignment between needles. The maximum racking amount is limited and is usually at least 2.



rack = 0 aligns `b0` with `f0` rack = −1 aligns `b1` with `f0`

`split D N1 N2 CS`

Split. The *split* operation combines knit and transfer into one operation. Split is useful because it allows the machine to knit through a loop without losing the ability to access the loop in the future.

In the knitout specification, the split operation includes necessary details for both a knit and transfer – the direction **D** of yarn-carrier, needle-bed position N_1 of the source loops, needle-bed position N_2 of the split loop, and yarn-carrier set **CS** participating in the operation.

A few utility operations for finishing knitting and handling yarn include:

Drop. The instruction `drop` causes a needle to drop the loops it is carrying. This is identical to `knit` with no yarn.

```
drop N
```

Miss. The instruction `miss` causes the specified yarn-carrier set to move *as if* a loop is being added to the specified needle (without actuating the needle and adding the loop).

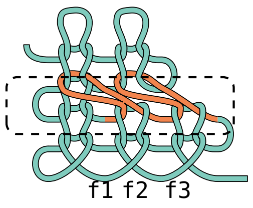
```
miss D N CS
```

In, Out. The instructions `in` and `out` add and remove active yarns. When a yarn is removed, it is cut and the connection between it and its last stitch is broken. Some machines may support an additional yarn handling helper device that helps hold the yarn during insertion and removal. This is explicitly supported in knitout using the **inhook**, **releasehook** and **outhook** commands.

```
 in[hook]   CS
out[hook]   CS
```

The following knitout snippet decreases the width of a 3-loop wide sheet constructed on the front-bed using yarn A by moving the right-most loop over to the left by one:

```
...
knit - f3 A
knit - f2 A
knit - f1 A
xfer f3 b3
xfer f2 b2
rack -1
xfer b3 f2
xfer b2 f1
knit + f1 A
knit + f2 A
...
```



2.5 *Miscellaneous*

A hand knitter (or anyone who owns a knit sweater) would realize that our discussion so far did not talk about colors or textures such as those seen in fair-isle style sweaters, ribs using purl stitches seen on sock cuffs, or other texture effects such as laces and cables.

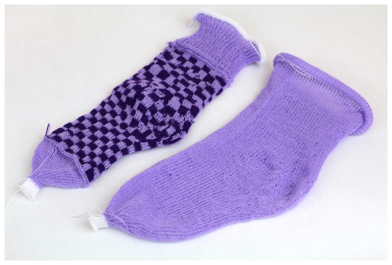


Figure 2.12: Socks with fair-isle color work and ribs on the cuffs

Knitting machines are set up to have multiple yarn carriers, each of which can feed yarn of a different color or material. Using different yarns in a pattern, various forms of colorwork can be introduced as seen in figure 2.12. Apart from color, yarns of different materials materials (e.g., elastic, conductive, and heat-sensitive) can be embedded in the knit object. The beanie shown on the right in figure 2.13 has a few courses of conductive yarn used to power LEDs.

By transferring loops in a pattern, various texture effects can be introduced. For example, by pairing decreases and increases carefully, a lace fabric can be created. Cables can be created by transposing a few loops in the pattern giving a characteristic textural effect as seen in the hand-warmer example.

A sheet knit on the front bed has a distinct appearance on the front-side and back-side. The front (out) side has a characteristic *v* shape and the back (in) side has a - dashed shape (called a 'purl' stitch in hand-knitting and a back knit in machine knitting). Knits and Purls are commonly used in garments for not just texture but also structural elements. Ribs – alternating columns of knits and purls – are more elastic and usually found on cuffs and socks. This change in fabric behaviour is because a knit loop does not lie flat. It curls one way along its legs or sides and in the opposite way along its neck. This saddle shape aggregates over the surface – for example, an all-knit sheet curls strongly both in opposite directions along the course direction and wale direction just like a single knit stitch does. Because purl stitches are essentially knit stitches facing the other way, they curl in the opposite direction. A careful combination of knits and purls can in fact balance out or exaggerate the aggregate curling behavior of the fabric. Combining knits and purls along with shaping gives rise to a rich space of knit textures. Figure 2.15 illustrates a number of ways to shape and texture a knit tube.

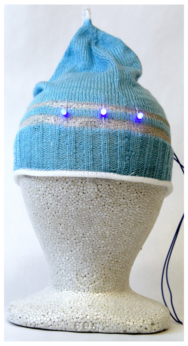


Figure 2.13: Using conductive yarn for LEDs

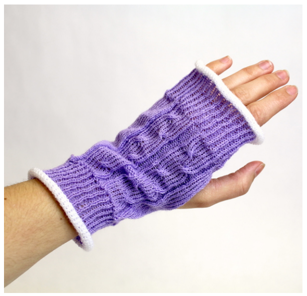


Figure 2.14: A handwarmer with cable textures.

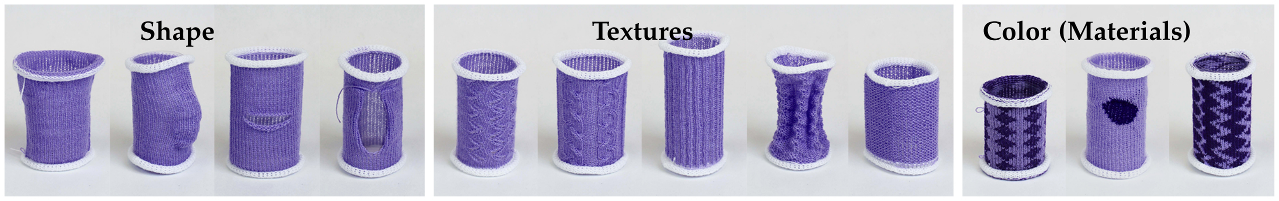


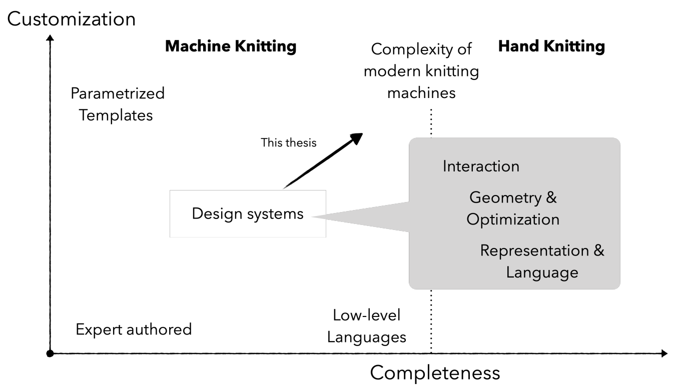
Figure 2.15: Knit shapes can be edited in various ways. Its shape can be changed by increases (decreases) and short-rows. Textures can be edited by using increases and decreases to create lace work, cables and knit-purl variations. Multiple materials can be introduced to edit the structure in various techniques like plating, intarsia and fairisle style knitting

3
Background

Knitting and other forms of fabric craft have a rich and long history. Postrel [2020] describes how fabrics have influenced human civilization in every step of the way. Mechanized Jacquard looms and knitting frames have played a key role in the industrial revolution, even paving the way for modern computing. Among the far reaching implications that the industrial revolution has had in almost every facet of our modern lives, it has certainly changed the way clothing and fabric is made. Providing a comprehensive background of this space is almost impossible. I therefore attempt to center this chapter around ideas that have influenced this thesis across fabrication, geometry processing and graphics.

3.1 Knitting

The chart below visualizes the problem space around knitting along two axes that the work in this thesis intends to make progress along: completeness and customization.



Pattern representations can be measured based on how much of the design space can be described using the representation. Low-level languages are likely to be complete – capable of representing *any* knitting pattern conceivable. However, they may not be directly amenable to high-level interpretation and design, required for customization. A goal of this thesis is to enable design systems that can represent a rich subset of the design space while also support high-level interaction and expressivity. Many of the sub-problems that arise in pursuing this can often be cast as a problem of identifying useful representations and data-structures, applying robust geometric algorithms, and, designing optimization and interaction based solutions.

3.1.1 Low-Level Languages and Representations

The knitout language proposed by McCann [2017] and described in the introduction as well as languages supported by industrial knitting CAD systems such as KnitPaint by Shima Seiki [2011] and M1 Plus by Stoll [2011] can be classified as low-level languages. These representations can be used to describe any machine-knittable pattern but require specifying details at the stitch and machine needle level. An expert knit programmer may hand author custom patterns at the stitch level using these systems. To provide high-level control and support common designs at scale, these systems also support parametric templates for garments such as sweaters and gloves. Libraries of textures are also maintained and can be applied to patterns and further edited (Shima Seiki [2019], Soft Byte Ltd. [1999]). Guidebooks of advanced techniques do exist that can assist with this process (Underwood [2009]).

Modern knitting machines are constrained in the way they can manipulate yarn. Therefore, machine patterns constitute only a subset of all knitting patterns. De Dillmont [1900] provides an excellent compilation of all forms of needle-work (including knitting patterns) in their encyclopedia. Human knitters are dexterous and able to form complex stitches. Indeed, Belcastro [2009] showed that a 2D surface of any topology can be hand knit. Although any surface can be hand knit, understanding the structure of knitting is an interesting and challenging problem. Topologists have looked at formalizing knit structures using knot theory. Grishanov et al. [2009a] studied textile structures like knitting and weaving as knots and links on a torus to capture their periodic nature. Markande and Matsumoto [2019] presented a topological framework to describe knit swatches, viewing knit stitches as knots on a thickened torus with an algebra to join them and make a fabric.

Hand-knitters also use low-level languages such as Knitspeak to represent hand-knitting patterns. Battell [2016] presented a domain-specific language called Purl, syntactically similar to knitspeak that can verify hand-knitting programs. Hand knitting books often focus on textures, coming up with instructions for knitting complex shapes can still be challenging.

3.1.2 3D Modeling and Pattern Generation

Knit patterns have a strong row and column structure formed by the courses and wales. Describing a knit pattern on a surface can be viewed as a parametrization problem over these orthogonal directions. Singularities in the surface parametrization can be interpreted as knit structures that introduce increases, decreases or short-rows. This setup naturally lends its way to using directions as inputs when designing a pattern, particularly for seamless machine knitting. Given a 3D shape that needs to be represented as a knitting pattern, direction constraints can be imposed to suggest courses (rows) or wales (columns). The constraints of machine knitting, however, require that these directions be acyclic i.e. the underlying direction fields must be curl-free (i.e., gradient of a scalar field). Field-guided remeshing and vector field processing have been successfully used to capture important shape properties for fabrication in general, such as alignment to principal curvature directions (Pottmann et al. [2008]), stress fields (Madan et al. [2020]), and general directional preferences of the user (Jakob et al. [2015]).

Although yarn is continuous, machines manipulate the yarn in terms of discrete and highly structured loops. Stitch patterns can therefore be modeled as a charts (a discretization of the parametric surface), graphs (treating stitches as nodes and their yarn connectivity as edges) or polygon meshes (representing stitches as vertices or faces of the surface). It is worth noting that a knit loop does not lie flat. The action of pulling the loop through another loop turns each loop into a saddle-like structure (Peirce [1937]). This can lead to interesting 3D surface structures and can even be used to model auxetic meta-materials (Hu et al. [2011]). Cirio et al. [2015] model knit structures as discrete hexagonal elements with hinge angles to simulate 3D knitting patterns very effectively. Instead of using hinge angles, Wadekar et al. [2020] use a helicoid structure to support the curved shape of knit loops. Kapllani et al. [2021] use a graph structure to record the topology of machine knitting patterns.

Yuksel et al. [2012] represent knitting patterns as a quad-dominant "stitch" mesh with a focus on visualization and simulation. To automatically come up with a stitch mesh from a 3D model, Wu et al. [2018] proposed a quad-meshing approach using a 2-RoSy field for guidance. Given that a general 2-RoSy field might not be a gradient field, to turn these into (hand)knittable structures, seams are introduced with mismatch faces.

Many field aligned quad-meshing algorithms conform to the surface shape well but individual quads can be differently sized with errors smoothly distributed over all the faces. When the number of stitch shapes that can be used to represent each face in the result is limited, this can lead to discretization errors that accumulate poorly. Pottmann et al. [2015] describe a similar challenge in paneling for architecture with a limited tile set and propose to interleave a continuous shape optimization step with a discrete panel optimization step that encodes the relevant fabrication constraints. Modeling stitch selection and meshing using such an interleaved setup would be an interesting approach to texturing 3D knitting patterns. Liu and Jacobson [2019] show that solving an as-rigid-as-possible energy minimization problem with an l1-regularization term can be used to generate cubic shapes. Using such a regularization term to optimizing meshing algorithms might be an interesting way to deal with the error accumulation problem in knitting.

A related quad-meshing approach, especially useful for fully-fashioned machine knitting, is to enable control of seams in knit objects directly. Hierarchical quad meshing ideas based on refining a base mesh – where the base mesh can be driven by features (either automatically detected as creases or by user input) may be interesting ways to support editing and patterning (Campen et al. [2012]). Hierarchical structures may also be useful in imposing additional constraints such as symmetry. Igarashi et al. [2008a,b] presented a design assistant that semi-automatically creates a knitting pattern from a 3D model by segmenting the shape into tube-like regions and covering the tubes with a winding strip and finding areas where increases or decreases are needed. The knit results could additionally be posed using wires. The end-user directly controls the high-level segmentation whereas the computational system identifies low-level stitch placement within each segment.

Although representing knitting patterns in 3D and generating patterns from 3D shapes is useful, the physical properties of yarn (especially when different textures are applied) is not uniform. This often means that the geometry of the 3D shape cannot be directly turned into a knitting pattern. Yuksel et al. [2012] used ideas from sub-division and adaptive remeshing to deal with texture variations. Liu et al. [2021] modify the underlying surface shape from which the pattern is created directly to account for texture variations, similar to Skouras et al. [2012]'s approach for balloon design.

Ideas from image manipulation in graphics have also been useful for manipulating textures and patterns. Hofmann et al. [2019] have looked at data-driven approaches for manipulating textures and introduced a system to parse hand-knitting patterns and combine them with seam-carving (Avidan and Shamir [2007]).

For hand-knitting, generating instructions from high-level pattern representations (such as a graph or mesh) is often simple. The constraints of machine knitting makes instruction generation more challenging and design systems for machine knitting need to account for it.

3.1.3 *Design Systems for Machine Knitting*

Traditional design tools work in the construction space of the machine – requiring users to figure out the construction location (at which needle must a stitch be created) and construction order of stitches (on which pass must this be created) at the same time as they determine stitch and connectivity. Further, it is the designer's responsibility to ensure that stitches and transfer instructions are encoded appropriately and efficiently.

Meißner and Eberhardt [1998] proposed one of the earliest approaches for visualization of machine knitted structures. Beginning with machine knitting data (WKT format from Stoll knitting machines) – stitch information and material information were recorded. A particle simulation approach was used to then simulate and visualize these patterns. Over the past few years, there has been a renewed interest in fabricating with machine knitting. Popescu et al. [2018] described a system that automatically generates a knit representation for topologically-disc-shaped patches, which are later connected manually. However, many intermediate steps including patch segmentation and machine layout remain manual in their system. Jones et al. [2020] introduce a system to support patch-level pattern editing while maintaining low-level knittability constraints Jones et al. [2020]. Recently, Nader et al. [2021] presented a graph rewriting based approach for supporting 3D knitting of meshes with textures. Kaspar et al. [2019b] present a system that learns machine knitting instructions by curating a dataset of KnitPaint programs and images of the associated fabricated results. Kaspar et al. [2019a, 2021] presented an interactive design system in the construction space coupled with techniques to compose textures for surface patterning and force-layout based embedding. More recently, they presented an approach to turn cut-and-sew patterns into seamless machine knitting patterns. Albaugh et al. [2019, 2021] have also looked at novel structures that can be manufactured with machine knitting such as integrated tendons that can be used for actuating soft objects and spacer fabrics. In addition to making feasible knitting patterns, researchers are also looking at creating efficient patterns, validating patterns for correctness and characterizing efficiency for machine knitting (Lin and McCann [2021], Lin et al. [2018]).

Machine manufactures continue to innovate hardware systems for knitting techniques, speed and robustness (ITMA [2019]). Additionally, Kickstarter initiatives such as Kniterate are offering low-cost consumer grade machines with many of shaping capabilities of industrial machines (Rubio et al. [2017]). Finally, apparel designers and retailers are keen on customized, on-demand knitwear and allow customers to personalize certain aspects of expert-created patterns for design and fit (Adidas [2019], Ministry of Supply, Unmade).

3.2 *Clothing Design*

In contrast to constructing fabric from the yarn-level, most clothing design work in graphics has focused on the "cut-and-sew" paradigm, where clothing is sewn together from multiple panels cut from flat fabric; with many contributions in simulation (Carignan et al. [1992], House and Breen [2000], Volino and Magnenat-Thalmann [2000], Volino et al. [2009]) and interactive and intuitive interfaces (Decaudin et al. [2006], Mori and Igarashi [2007]). Wu et al. [2021] have looked at adding seams to automatic machine knitting techniques to ensure wearability.

One of the difficulties in cut-and-sew clothing is fitting 3D models, which is done by placing specific shaping features like darts and folds (Li et al. [2018], Turquin et al. [2007], Umetani et al. [2011], Wang [2018]) or by combining and adjusting patterns in a physically meaningful manner (Bartle et al. [2016]). Further, data-driven approaches have been used for parsing sewing garments into 3D draped forms as well as for exploring the multi-modal design space of body shapes, textures, and 2D patterns (Berthouzoz et al. [2013], Wang et al. [2018]).

Beyond traditional clothing, knitting has also been an effective tool for building responsive sensors and smart wearables (Farringdon et al. [1999], Luo et al. [2021], Ou et al. [2019a,b]).

3.3 *Other Soft Fabrication Systems*

3D printing is often thought of as process to construct rigid structures. However, by using the appropriate micro-structures, fabric-like drapes can be achieved (Peleg [2018]). Cuts can be introduced into rigid metal structures to shape complex curved structures (Konaković et al. [2016], Malomo et al. [2018]). Apart from micro-structure based manipulation of otherwise rigid structures, 3D printing has also been used to manipulate elastic fabric by overlaying rigid patterns over fabric laid out under tension which generates a 3D form on release (Guseinov et al. [2017], Pérez et al. [2017]).

Like knitting, weaving is another common fabric-making technique. Recently, researchers introduced a sketching-based design system that let designers construct 3D woven structures by drawing weft yarns, tiling these designs and using simulation to guide designs (Harvey et al. [2019]). Akleman et al. [2009] showed how graph rotation systems can be used to turn polygon meshes into plain-weave structures. Vekhter et al. [2019] introduced a computational approach based on geodesic foliations on a surface to cover it with tri-axial weave structures that can be manually fabricated.

Felting is another classic soft fabrication technique where pieces of wool are bonded together by piercing with a barbed needle that causes fibers of different orientation to tangle and hold shape. Hudson [2014] introduced a felt printer similar to a 3D printer but using felting yarn as a filament and barbed needle to connect felted layers.

3.4 *Simulation and Rendering*

Researchers have been interested in understanding and modeling the structure of knits across various communities – textiles and material science (Leaf [1960], Leaf and Glaskin [1955], Peirce [1937]), geometry and topology(Grishanov et al. [2009b]), graphics and simulation (Kaldor et al. [2008], Yuksel et al. [2012]). Yarn-level simulation methods can produce realistic deformations of knitted structures.

Recent works have focused on efficient methods to simulate yarn-level details, including combining Lagrangian and Eulerian approaches (Sueda et al. [2011]), applying reduced order methods (Cirio et al. [2017, 2014, 2015]), and modelling anisotropic deformations using the material point method (Jiang et al. [2017]). Leaf et al. [2018] demonstrated interactive yarn simulation for periodic pattern design using GPU computation. Another area of active interest is characterizing the mechanical properties such as the elasticity or auxetic nature of knit fabric (Hu et al. [2011], Knittel et al. [2015]).

Researchers have been working on photorealistic knit fabric rendering for over a decade (Groller et al. [1995], Gröller et al. [1996]). Approaches include representing the geometric complexity of knit structures with volumetric approximations (Chen et al. [2003], Xu et al. [2001]), CT scan data (Zhao et al. [2011]), and procedural functions (Zhao et al. [2016a]); and rendering the data with the radiative transfer framework (Jakob et al. [2010]), the SGGX microflake distribution (Zhao et al. [2016b]), and data-driven approaches (Aliaga et al. [2017], Khungurn et al. [2015]). For interactive speeds and reduced memory, details can be created on-the-fly (Lopez-Moreno et al. [2015], Luan et al. [2017], Wu and Yuksel [2017a,b]).

4
What Can Be Machine Knit ?

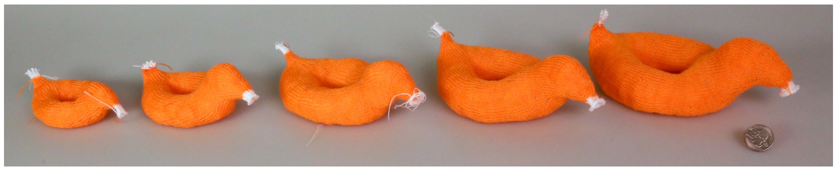


Figure 4.1: A row of knit ducks of different sizes created directly from a 3D mesh and a scaling factor. This pattern was constructed seamlessly in one piece by the knitting machine and stuffed in post-process.

What can be made on a knitting machine? Now that we know the machine operations in knitout, one answer is: any sequence of knitout operations can be performed on the machine. In reality, an arbitrary sequence of instructions is more likely to result in a tangled mess of yarn than a meaningful result. What we really mean is – what sort of shapes can be produced on the machine?

Understanding the class of surface shapes that can be machine knit is key to our goal of separating the high-level design space from machine operations. Without such a classification, any separation will be meaningless – letting us create designs that cannot be executed on the machine. We have seen three styles of knitting machines – linear flat bed machines, circular knitting machines and two-bed (and its multi-layer variants) machines. It is clear that single bed knitting machines can make topological discs (sheets with short-rows) and circular machines can additionally make a tube that can be bent with short-rows. Here, we will focus on the design space of the general two-bed machine. Figure 4.1 shows a toroidal duck knit in one-piece on a two-bed knitting machine with two layers. What sort of layer setup is needed to construct an arbitrary surface if it can be constructed at all?

Given a knit object, we say that it represents a *surface* if the knit loops uniformly and densely cover the surface shape and locally, the loops are arranged like a sheet almost everywhere. The thickness of this sheet and the thickness of the yarn forming the loops are negligible in relation to the size of the surface. An example of an everyday object that falls in this category is a knit T-shirt. A knit T-shirt might be crumpled and laid out in a complicated way at any point in time, but it can always be smoothed out to a surface. Not all knit objects need to be a surface as described here. A multi-layered dish cloth for example, might be constructed by layering sheets that may themselves be knit surfaces and connect them with yarn by knitting, sewing, or gluing them in various ways such that it fills a volume. In chapter 8 we discuss how ideas presented in this thesis might be expanded to handle volumetric structures, but largely this work focuses on representing and fabricating knitted surfaces.

Knitting a Surface Consider an arbitrary surface:

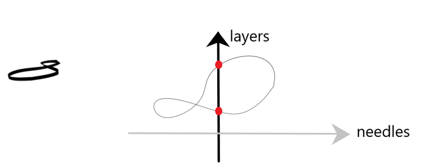
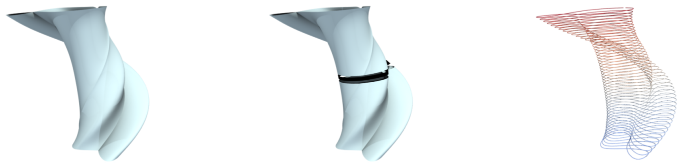


Figure 4.2: If an arbitrary surface can be sliced such that each slice can be placed on the multi-layer machine in a reasonable way, the surface can be stacked and knit.The surface is sliced along its height, shown by contours colored blue to red.



An iso-contour of the height function of this surface is a planar curve and can be laid out on a continuous version of a multi-layer machine – with infinite needles and layers as shown in figure 4.2. The curve can be constructed by knit loops connected by yarn on the needle plane. Discretizing the curve (and the machine) can be viewed as placing loops (on needles) at a uniform separation, allowing the continuous yarn between loops to form the curve between two points on the curve as shown in figure 4.3. Any part of the curve that is orthogonal to the needles, can be perturbed or be followed by the stretchy yarn between loops so that any line in the layer plane orthogonal to the needle direction intersects the surface in only finitely many loop locations. Any two consecutive slices can be *connected* by knitting through loops of the previous slice when constructing the next slice. Intuitively, if a surface can be generated in one piece by stacking and connecting slices (each of which can be held on the machine) – it is machine knittable. More specifically, we would like to:

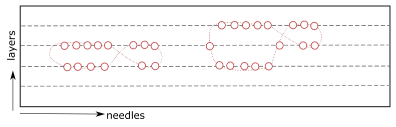


Figure 4.3: Although more layers can be used, a figure-8 like shape can be placed on two layers. Here discrete loops are shown by a circle and the dotted line between them indicates yarn. The dashed lines show the layers.

Problem Statement 4.1:. Characterize the space of surfaces in $\mathbb{R}^3$ that can be constructed with machine knitting.

4.1 *Design Space of a (Multi-Layer) Machine*

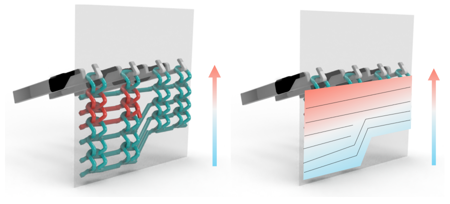


Figure 4.4: Assigning a time function $f : S \to \mathbb{R}$ on a fabricated surface S.

Machine knitting is an *additive* fabrication technique, creating the shape by adding one loop another, row after row. A scalar function can be assigned to each point on the fabricated surface, based on the time at which the surface was created. This implies that *any* machine knit surface can be associated with a scalar time function whose contours (at unit time intervals) provide the rows of loops (courses) that forms the knit surface.

On a (multi-layer) two-bed machine, these contours can appear as closed cycles[1] or open curves. Because the constituent loops can be moved around by `transfer` and `rack` operations, the length of the contours can vary. Because multiple cycles can be held on a two-bed machines at any time, at any time (iso-value), multiple disjoint iso-contours may exist.

[1] The construction of a tube proceeds in a helical fashion which means the rows are never strictly closed, but the surface contours can nevertheless be viewed as closed cycles

Two or more contours produced at time t_i may be merged at time t_{i+1} into a single contour by knitting through loops of all those previous loops with a single new row (chain or cycle) or loops.

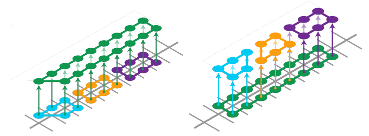


Figure 4.5: Merging and splitting cycles

A contour produced at time t_i may split into multiple contours at time t_{i+1} by using a new yarn that creates multiple new cycles through the previous row.

Although a cycle can be *rotated* and *translated* on the machine, once laid out, it cannot be reflected inside-out – the orientation of the surface cannot be changed. If a closed manifold surface is constructed on the machine, it must be oriented. Each contour must have been constructed on some layer(s) of the machine and over some needle(s) in those layers. Each contour c can be associated with a list of layers $l(c)$ and needles $n(c)$. Two disjoint iso-contours that appear at the same iso-value cannot share needles from the same layers – since there is no operation to separate loops that appear on the same needle location.

Further, two cycles that share a layer cannot pass across each other – there aren't enough needles to execute such an operation because in practice, the amount by which the layers can translate is bounded by the racking limit of the machine. Therefore, the relative ordering of a sequence of cycles on the machine bed cannot be changed within a layer but a cycle can be moved from one layer to another.

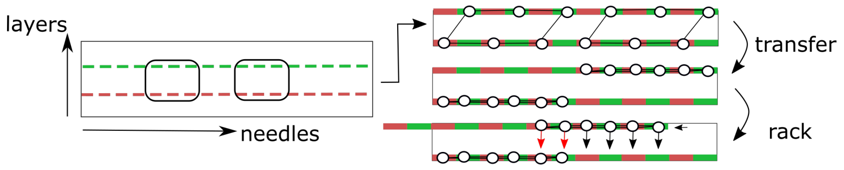


Figure 4.6: Although two cycles can be held at the same time with two layers, their positions cannot be swapped unless the racking limit of the machine is greater than the width of the cycle. Here, two cycles are shown side by side, after transferring to collapse the cycles and racking them, there are no free needles available to expand the cycle and repeat this procedure.

These constraints on the shapes and their transformations suggest a way to classify any surface that can be fabricated on our multi-layer machine model with some fixed number of layers L. For any surface $S \in \mathbb{R}$ made with such a machine, it is true by construction that:

1. A scalar function can be defined on the surface of the knit structure by recording for each stitch the time at which its row appeared on the knitting machine i.e., the contours of this function are curves

and describe the rows of the knit object. A similar function can be defined on the surface for recording needle on which each stitch was constructed and the layer on which it was constructed.

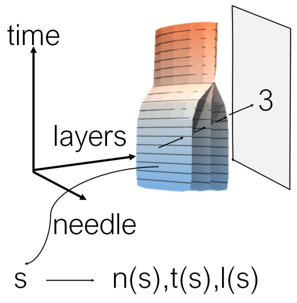
2. Each contour of the time function can be identified by contracting it into a single point. This process generates a *skeleton* for the surface (similar to the Reeb graph (Reeb [1946])). The evolution of these contracted contours through time produce edges of this skeleton. Critical points where contours merge or split appear as vertices of this skeleton.

3. The graph appears in time order, so the drawing must be upward (all the edges are oriented consistently with the time function) and is directed-acyclic (no loops can be used from the future).

4. From the physical setup of the machine, there must exist two orthogonal directions $\vec{l}$ and $\vec{n}$ (for layers and needles). Any two contours of a level set of the time function that share the same set of layers must necessarily be held on different needles to avoid intersections. Knit loops are held on needles, yarn between loops can connect loops within the same layer or along different layers.

5. The embedding of the surface in the construction space of the machine defined by the time $t : S \to \mathbb{R}$, layer $l : S \to \mathbb{R}$ and needle $n : S \to \mathbb{R}$ functions, $f(x \in S) \mapsto (n(x), t(x), l(x))$ is topologically equivalent (isotopic) to the surface S.

   A projection of the surface $P(x \in S) \mapsto (n(x), t(x))$ along the layers has at most L points that coincide. For example, the figure on the right requires 3 layers for construction.

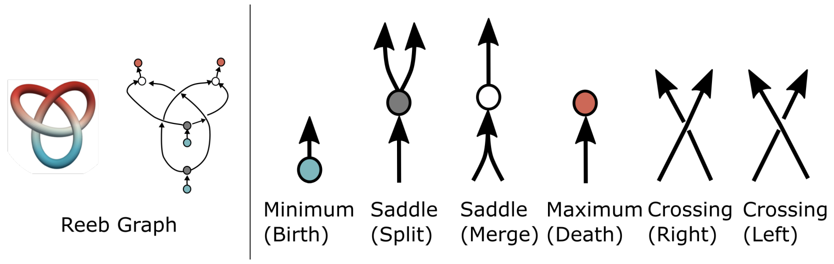


In other words:

A two-bed (multi-layer) continuous knitting machine that makes infinitesimally small stitches can construct a 2D surface $\mathcal{S}$ iff there exists functions $t, n, l : \mathbb{R}^3 \to \mathbb{R}$, with an isotopy between S and $f(x) \mapsto (n(x), t(x), l(x))$ for $f : S \to \mathbb{R}^3$ and projection $p : S \to \mathbb{R}^2$ where $p(x) \mapsto (n(x), t(x))$ and $\forall x \in p(S)$, $|\{y | p(y) = x\}|$ is finite.

Although knittability of a surface can be shown with many layers, in practice layers are achieved at the cost of needle resolution. Further, any contour can be held flattened on just two layers (assuming yarn is stretchy). In practice, an embedding that minimizes the number of layers when possible is desirable.

Often, one is only interested in 'nice' manifold surfaces that locally appear like a sheet with no T-junctions. Here we assume these manifold surfaces form a single connected component (e.g., not a concentric collection of 2-spheres). From Morse theory, there exist a height function for any manifold surface $\mathcal{M}$ that with minor perturbations only has binary splits and merges (Edelsbrunner and Harer [2010]). This height function can be viewed as the time function for ordering slices. When viewed as a projection on a plane parallel to the time direction, the the Reeb graph of $\mathcal{M}$ can have the only following events:



Events of interest on a projection of the Reeb graph.

By perturbing the time function, one can ensure that any time exactly one event occurs. Notice that all contours will appear as simple cycles or simple curves and can be laid out without any intersection adjacent to each other[2]. The Reeb graph is directed (edges are directed by time) and is acyclic. Merging and splitting events can occur on a single layer for sheets or two layers for tubes. Crossing events require an additional layer since at the crossing point, two tubes must be held simultaneously and disjointly on the machine.

[2] The edges of this graph represent cylindrical components, that do not intersect or pass through each other since that would introduce a self-intersecting surface

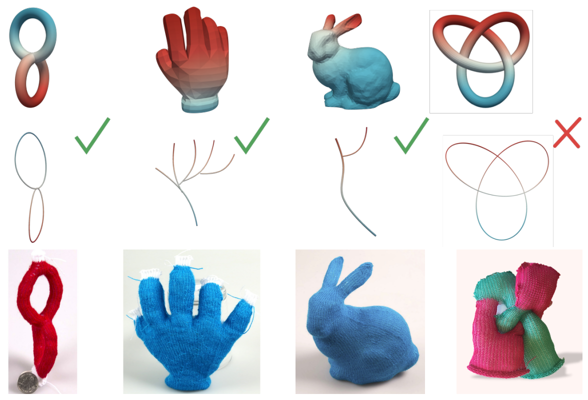
Although a single projection of this graph to a plane might enforce an arbitrary left-to-right ordering of cycles on the machine at critical points – cycles can always be rotated, either before the critical point or after, ensuring that any left-right ordering of cycles can be achieved using binary splits and merges.

For machines with exactly two layers (standard knitting setup) and tube-like surfaces, no crossing events can appear. This setup is often more robust because the fabric layers do not pass across each other between the beds. For a machine with four layers, any crossing can be executed by placing the cycles associated with the crossing edges on two different sets of layers. At any instance in time, only one crossing event appears, hence additional layers are not necessary. These constraints can be formalized as: [3]

[3] Nested manifold surfaces can be constructed by using as many layers as the nesting complexity

A continuous two-bed four-layer knitting machine that makes infinitesimally small stitches can construct any oriented 2D manifold $\mathcal{M}$. A two-layer knitting machine can construct an oriented 2D manifold $\mathcal{M}$ iff there exists a (Morse) function f whose Reeb graph has an upward planar embedding.

Some examples of what can and cannot be fabricated in *one piece* on a *two-layer* (i.e., standard industrial) machine[4]:



[4] The trefoil knot has an upward drawing with edges that pairwise intersect on the layer plane and would require 4 layers to construct as each edge represents a cyclic contour (tubular segment).

This characterization based on the topology of the surface is useful in identifying if a surface is machine knittable at all with a fixed number of layers, irrespective of the discretization used to come up with stitches and loops.

Geometric properties The discussion so far, focused mainly on the topology of the surface being constructed. To meaningfully fabricate a surface metric properties are also important. The distance between points on the surface needs to match the distance between loops on the constructed surface for all points on the surface. Consider a time function used to generate a slice of the object. The distance along the surface between two points on consecutive slice (when moving along a straight geodesic path from a point on the lower slice) might be different. When being constructed by knit loops, only a loop-height worth of distance can be covered by any contour. The slice can therefore be further subdivided into regions that can be covered by a fixed height along the surface. These sub-patches can be constructed using *short-rows* on the machine. In chapter 6, I describe an incremental remeshing algorithm that uses this idea to explicitly convert a 3D mesh into a knitting pattern.

4.2 *Physical Constraints*

A goal of this thesis is to enable a pipeline that supports *fabrication* of machine-knit objects. This involves translating idealized assumptions like zero-volume infinitely-stretchable yarn strands, unbounded curvature changes, infinitely strong needles, and long needle-beds into constraints that lead to robust and repeatable performance. Identifying these physical constraints, in turn, involves calibration, tuning and empirical analysis.

Maintaining Yarn Slack During knitting, loops may be moved around for shaping or scheduling other loops. During these operations, if the yarn breaks, the object being constructed is compromised because this failure cascades over all dependent loops that are no longer stabilized. The extent by which yarn can stretch depends on its physical characteristics and may differ based on yarn type and stitch size settings used on the machine.

For two loops constructed in sequence i.e., yarn-wise adjacent, the amount of yarn between them can be measured in terms of the needles between them. We refer to this as the "slack" associated between those two loops as illustrated in figure 4.7. Bringing these two loops closer is safe, separating them by a distance much greater than their slack is likely to cause yarn breakage.

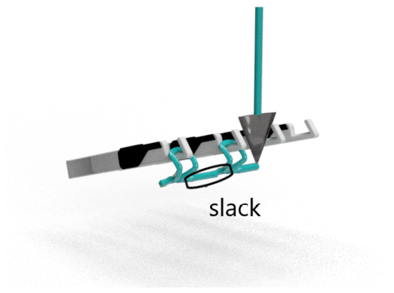


Figure 4.7: The length of the yarn between two loops at the time of construction is referred to as the *slack* between them. When loops are moved by distance greater than their slack, yarn can break.

Property 4.1: Slack. For yarn-wise connected loops i and j, constructed at needles n_i and n_j,

$$d(i,j) \leq k \cdot d(n_i, n_j)$$

at all times, where $k \geq 1$ may be experimentally determined and d is the distance between them in needle spacing.

$$\begin{aligned} d(n_i, n_j) &= |n_i - n_j| \text{ if i and j are on the same bed} \\ &= |n_i - n_j| + 1 \text{ otherwise} \end{aligned}$$

In our system, we set $k = 1$ for any transfer planning in general and allow $k = 2$ for increase shaping. The unit distance between two adjacent loops need not be one, especially in the context of multi-layer systems. In general, the distance between two loops is L for an L-layer machine emulated on a two-bed machine. The factors introduced by slack constraints are applied with respect to the unit distance accounting for layers[5].

This *slack constraint* can have consequences for patterning algorithms in multiple ways:

Limited increase Before the width of a row can be increased, a gap needs to be introduced by moving a few loops. This means that loops must be able to stretch at least by one unit (for multi-layer setups this might be more than one needle) to effectively be able to create any width variation [6]. As suggested above, we limit this based on the yarn slack to a factor of 2, allowing the fabric to increase by at most twice

[5] For a 3 layer machine, to introduce a gap loops may have to move by 3 needles or 1 unit. Note that the length of the yarn between adjacent loops is also 3 needles in this case.
There might be other ways to add more slack by creating longer loops https://alessandrina.com/category/machine-knitting/long-stitches-and-loops/ , but once created the slack between two loops is fixed.

[6] Other than adding new loops to the edge of the fabric

its width from the *previous* row. The physical stretch limits of the yarn can therefore also influence the number of layers a pattern using the yarn can have in general.

Balanced Layout For maintaining seamless appearance, limiting the possible layouts of a cycle and avoiding special cases, we assume cycles generally appear in a balanced fashion – the number of loops on the front bed(or layer) and the number of loops on the back bed(or layer) differ by at most 1 (to accommodate odd numbered loops).

During splitting and merging, it is important that this balance is maintained.

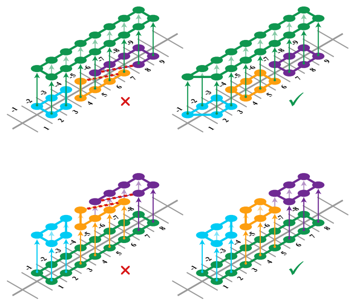


Figure 4.8: When cycles split or merge, it must be possible to lay them out in a balanced manner

Maintaining Needle Capacity The hook on each needle can hold a limited number of loops simultaneously reliably. This limit needs to be determined based on the hook size (usually a function of the number of needles per inch on the machine) and the diameter of the yarn. This limit influences how sharply the width of a fabric can decrease since decreasing involves overlapping loops to narrow the width. Based on our machine (and convenient symmetry with the increase limits) we set this to 2 for all our automatic patterning algorithms.

Layer Friction Although it is possible to emulate a multi-layer machine on a two-bed machine, this introduces two challenges. The first is the loss in needle resolution when multiple layers need to be maintained. The second is that the friction between multiple layers of overlapping fabric can lead to non-robust fabrication performance (depending on the yarn). It is therefore important that scheduling algorithms minimize the number of layers when possible and use multiple layers only when the underlying topology cannot be constructed with a simpler layer setup.

With these design and physical constraints in mind, I will next discuss data-structures to represent discrete knitting patterns.

5
Representing and Editing Machine Knitting Patterns

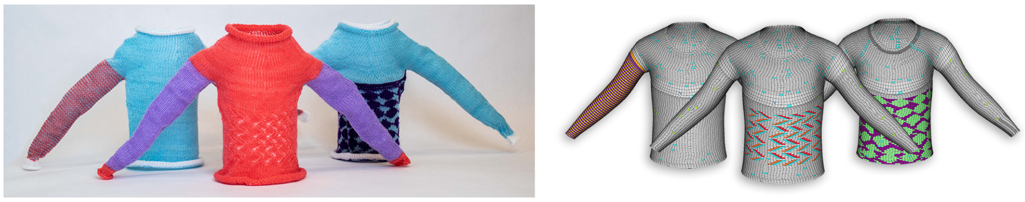


Figure 5.1: Beginning with an identical base pattern representation, quick edits (in this case under fifteen minutes each) can be performed to generate multiple variations.

A 3D mesh can be tested for machine knittability by using guiding scalar functions and verifying that an appropriate upward embedding exists for its Reeb graph. However, in order to construct a knit version of the shape, we need some way to turn a shape into loops and capture their dependencies. Additionally, it must be easy to construct this representation from 3D shapes, track machine-knitting constraints and finally convert it into knitting code. In this chapter, I will describe a new datastructure – an augmented stitch mesh – to represent knit structures.

For a datastructure that represents a valid machine knittable pattern to be useful it should satisfy a few desirable properties:

Property 5.1: Complete.

- Any machine knittable pattern must have a valid representation.
- It must be easy to verify that the pattern is consistent and machine-knittable.
- The data structure must represent the knitting pattern completely – it must encode all the yarn-wise and loop-wise dependencies of the stitches, track the order between loops, and maintain their adjacency information.

Property 5.2: Editable.

- The representation must be easy to visualize (along with its 3D embedding) and intuitive to edit with simple operations (i.e., operations over local stitch neighbourhoods).

- These edit operations must be able to span the space of valid patterns.

Property 5.3: Independent.

- The data structure should not encode or depend on physical assignments such as needle location or yarn carrier assignments.
- The data structure is not tied to a particular hardware architecture.

The key structure for representing a knitting pattern involves the loop-wise (columns or wales) and yarn-wise (rows or courses) connectivity. The stitch mesh data structure, introduced by (Yuksel et al. [2012]), represents knitting patterns using faces to encode stitches and labelled edges that dictate connections between faces (either yarn-wise or loop-wise). The knit graph structure introduced in Narayanan et al. [2018] represents knitting patterns with nodes for loops and labelled, directed edges for their dependencies. The dual of this knit-graph, naturally leads to a stitch mesh style structure (although the directed edge information in the graph is not represented in the stitch mesh) as shown in figure 5.2

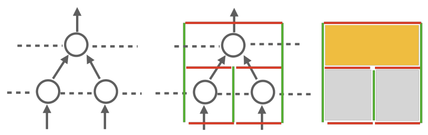
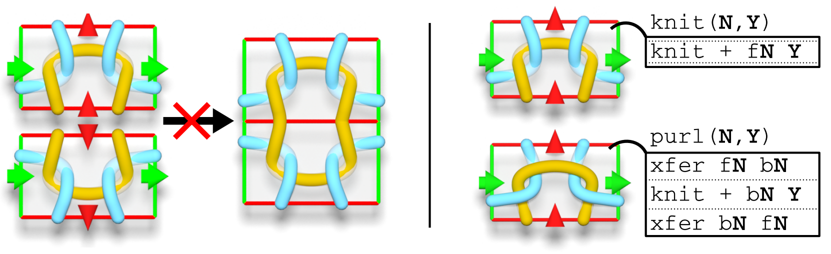


Figure 5.2: left: a snippet of a knit graph showing a decrease structure, middle: dual overlayed with colored edges, right: stitch mesh version – red edges indicate loop-wise connections and green edges indicate yarnwise connections. The yellow face with two incoming loop-wise connections can represent a 'decrease'.

The stitch mesh structure lets us build complex knitting patterns as a set of generalized Wang tiles (Wang [1961]). Faces can connect as long as edge labels (red loop-wise connections, green yarn-wise connections) match – which ensure that yarn-wise edges and loop-wise edges connect in reasonable ways. However, even with these constraints, one can produce structures that are not machine knittable because directional dependencies are not encoded. Second, machine instructions for constructing the stitch face need to be identified and may not exist for complex hand-knitting operations.

5.1 *The augmented stitch mesh*

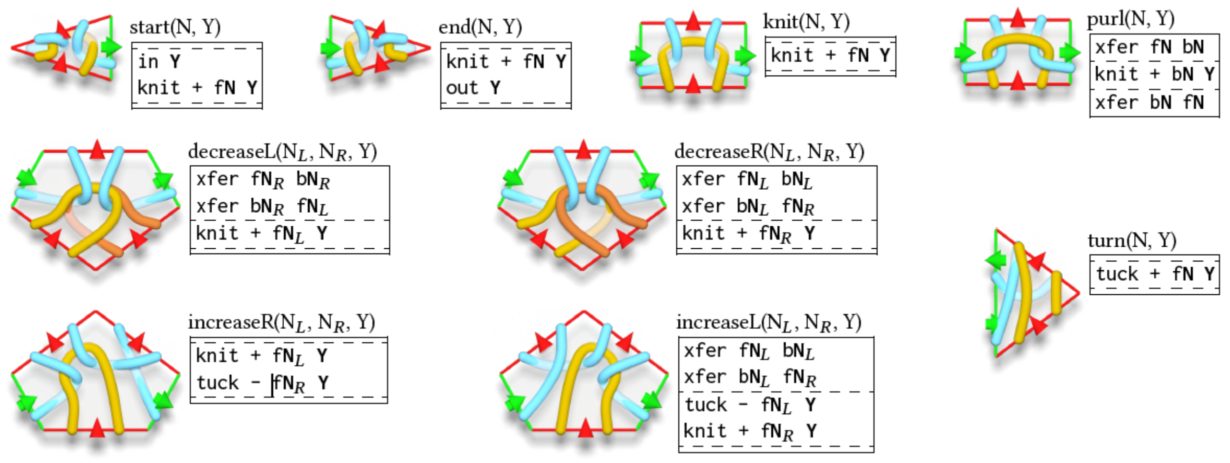
We augment the stitch mesh representation in two ways in order to support the editing of machine knitting programs: first, we add directed edge labels to track dependency information; second, we associate with every face type a knitting machine program that can construct the yarn-level topology on that face. Finally, we impose a constraint that the augmented stitch mesh must be directed-acyclic under the edge directions introduced.



```
knit(N,Y)
knit + fN Y

purl(N,Y)
xfer fN bN
knit + bN Y
xfer bN fN
```

Figure 5.3: Augmented stitch mesh faces have, *left*, directed edges to prevent locally un-knittable assembly; and, *right*, associated knitting programs.



```
start(N, Y)
in Y
knit + fN Y

end(N, Y)
knit + fN Y
out Y

knit(N, Y)
knit + fN Y

purl(N, Y)
xfer fN bN
knit + bN Y
xfer bN fN

decreaseL(N_L, N_R, Y)
xfer fN_R bN_R
xfer bN_R fN_L
knit + fN_L Y

decreaseR(N_L, N_R, Y)
xfer fN_L bN_L
xfer bN_L fN_R
knit + fN_R Y

turn(N, Y)
tuck + fN Y

increaseR(N_L, N_R, Y)
knit + fN_L Y
tuck - fN_R Y

increaseL(N_L, N_R, Y)
xfer fN_L bN_L
xfer bN_L fN_R
tuck - fN_L Y
knit + fN_R Y
```

Figure 5.4: Basic face types and their associated knitting code fragments. Faces with opposite yarn direction proceed similarly. Dashed lines indicate divisions between construction passes. Variants of these faces can be maintained for specific behaviour. For example, the output loop for decreaseL arrives on fN_L, and a variant decreaseR could have it arrive on fN_R.

Directed Edges The edge labels in the augmented stitch mesh capture dependencies between faces – *loop in* edges indicate that a loop is needed, while *loop out* edges indicate that a loop is produced; *yarn in* and *yarn out* give similar information about yarns. Any *in* edge may only connect to an *out* edge of the same type, and visa-versa. As shown in figure 5.3, these labels ensure that invalid closed links are not represented by the stitch mesh. A minor subtlety worth noting is that for machine knitting, the loop edges are associated with *needles* holding loops because two loops held on the same needle cannot be separated. Also, yarns and loops across faces are connected by edges that share the same label and have compatible directions, however there is no constraint that edges share vertices over these connections. This allows us to represent both manifold and non-manifold surfaces with the augmented stitch mesh structure. These directed edges induce a directed graph (like the knit graph) on the faces.

Face Programs Each face in our augmented stitch mesh data structure represents a fragment of a knitting program (specified in knitout) and configuration information that operates on the yarns and loops provided by its *in* edges in order to produce the yarns and loops indicated by its *out* edges (Figure 5.3) and the *layer* on which the face lies. That is, the edge labels provide a type signature – input and output loop and yarn counts – for a knitting program fragment to be executed on some layer. The layer and the configuration information is used by the scheduler to appropriately shuffle the loops held on the machine so that the code fragment is correctly interpreted. The basic face types provided by our system, along with pseudo-code for their knitting program fragments, are shown in Figure 5.4. Importantly, our system makes it easy to extend this list, since subsequent editing and layout operations depend on face edge labels, not on the referenced program. It is also easy to extend to polygon faces with an arbitrary number of edges. The face programs provide a template of the machine code that can be executed on any layer of the machine since each layer is in turn associated with needles on both the beds of the machine. The face program can include any operation in the knitout assembly language described in chapter 2 but it is required to leave the machine at a zero-racking state at the end of its execution.

Directed-Acyclic Constraints A stitch mesh with edge directions and face programs is a valid augmented stitch mesh only if it is directed acyclic under the directed edges. Physically, this avoids representing cycles along loop-wise edges – no loop can be consumed before it was created. Similarly there exists no cycles along yarn-wise edges – yarn cannot be connected to form a closed cycle on the machine.

Augmented Stitch Mesh Definition Concretely, the augmented stitch mesh is defined as $ASM = (M, G, C, L)$ consisting of mesh M, geometry library G, code library C and the total number of layers L.

The mesh $M = (\mathcal{F}, \mathcal{C})$ consists of a list of faces $\mathcal{F}$ and connections between faces $\mathcal{C}$.

Each face $f = ((v_1, .., v_n), c, g, l) \in \mathcal{F}$ consists of a counter-clockwise sequence of vertices $(v_1, ..., v_n)$ where $v_i \in \mathbb{R}^3$ that defines the face polygon, an associated code face $c \in C$, geometry face $g \in G$ and layer number $0 \leq l \leq L$.

f_a^x is used as a short hand to identify x associated with face a e.g., geometry face g associated with mesh face a : f_a^g

A connection $e = (f_a^i, f_b^j) \in \mathcal{C}$ represents an edge-to-edge connection between face a and b. The edge in face a participating in the connection is defined by (v_i, v_{i+1}) if $i > 0$ and (v_{i+1}, v_i) otherwise (the addition operator $+$ is modulo n).

For an oriented, manifold mesh, each edge connection (f_a^i, f_b^j) has $sign(i) \neq sign(j)$ and each face edge participates in at most one connection.

Each face in the code library $c = (key, (e_1, ..., e_n), instrs)$ consists of a descriptive identifier key, a counter-clockwise list of edges $(e_1, ..., e_n)$

and a list of knitout instructions *instrs*.

Each edge in the code face describes the resource associated with that edge as $e = (dir, type, loc, yarns)$, where *dir* describes the direction of the resource (incoming or outgoing), *type* describes the resource type (loop or yarn), *loc* describes a local bed-needle location and *yarn* describes the yarn identifier.

For f_a^c, $|(e_1, ..)| = |(v_1, ...)|$ and mesh edge f_a^i is described by edge e_i in c.

The instruction list in each code face is checked so that only resources on edges with incoming directions are consumed within each face, and all the resource edges with outgoing directions are produced with each face.

Similar to the code library, the geometry library describes yarn geometry within each face as spline paths and can be utilized for visualization or extracted for yarn-level simulation.[1]

[1] Our system does not explicitly *check* that the code face and associated geometry face are consistent i.e., the knitout instructions in the code face produce the same topology of yarns as in the geometry face. This would be a useful addition, where the geometry face library is created by the code face library directly, appropriately clustering all geometry that are identical (e.g., a rightwards going front knit and a leftwards back knit).

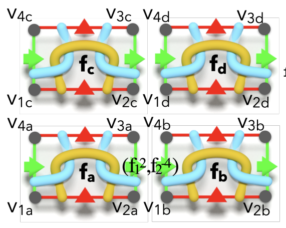
The geometry face $g \in G$ is described by $(key, (e_i, ...), (y_i, ...))$ where *key* is a descriptive identifier, $(e_i, ..)$ a counter-clockwise list of geometric edges and $(y_i....)$ is a list of yarn splines.

Each geometry edge $e = (v \in \mathbb{R}^2, dir, type)$ where v describes the 2D position of the source vertex in the edge, *dir* describes the resource direction and *type* describes the resource type.

Each yarn entry $y = (start, p_1, ..., end)$ lists a sequence of spline control points where $p_i \in \mathbb{R}^3$ and *start* and *end* lie on the edges.

For f_a^c and f_a^g, $|(v_1, ...)| = |(e_1, ...)|$, edge *type* and *dir* agree on the geometry and code face.

To maintain uniform density over the entire shape, the number of layers needed to represent a mesh is maintained as a part of the augmented stitch mesh *ASM*. An example is shown in figure 5.5. The edge directions from the template face types are used to verify that the mesh represents a directed-acyclic structure along yarns and loops.



$\mathbf{f_i} = ((v_1, v_2, v_3, v_4), purl_c, purl_g, l_0)$

```
face purl_c                face purl_g
   edge -l f0                 edge (-1,-1) -l
   edge -y f0.5 yA            edge (1,-1) +y
   edge +l f0                 edge (1,1) +l
   edge +y f-0.5 yA           edge (-1,1) -y
   code                       yarn [e3:0.75,0] (x1,y1,z1),…, [e2:0.725,0]
      ;;Carriers: yA          yarn [e1:0.25,0] (x2,y2,z2),…, [e2:0.275,0]
      xfer f0 b0              yarn [e0:0.275,0] (x3,y3,z3),…, [e0:0.725,0]
      knit - b0 A
      xfer b0 f0
```

$$\boldsymbol{M} = (F = (f_a, f_b, f_c, f_d), C = ((f_a^2, f_b^{-4}), (f_c^2, f_d^{-4}), (f_b^3, f_d^{-1}), (f_a^3, f_c^{-1})))$$

Figure 5.5: A small augmented stitch mesh that consists of four faces and four face-edge connections. Each face has an associated code and geometry structure.

Next, I describe why the augmented stitch mesh satisfies the desirable properties as a data-structure for machine knitting patterns.

A complete representation Each knitout operation can be associated with an augmented stitch mesh face (and corresponding basic face program) and edge labels specifying its dependencies. The stitch mesh built by aggregating these stitch programs provides a representation for a machine knittable pattern (with a potentially complicated embedding) (Property 5.1). It *is* possible that two programs represent the same pattern – for instance any number of back-and-forth transfers would lead to the same yarn structure or complex faces can be aggregated in multiple ways to provide equivalent but different stitch meshes. Similarly, any pattern may be compressed into a single stitch face as long as the face program accurately represents the complete pattern.

It is possible to represent patterns without an upward embedding that is intersection-free using the augmented stitch mesh. However this check can be done on the topology of the shape (all edges must be "upward") in some projection before construction and the mesh must not be self intersecting.

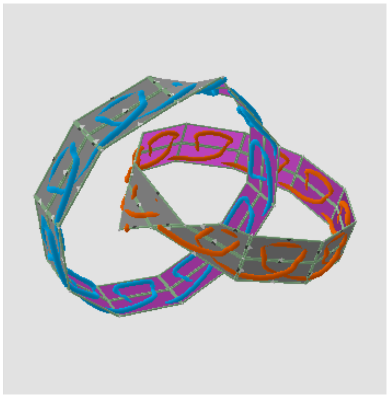


Figure 5.6: A smobj with linked cylinders as shown here does not have an embedding that is both intersection free and monotonic. Note that a surface with this topology can be constructed with a different stitch mesh.

A machine-independent representation The augmented stitch mesh tracks loop and yarn dependencies and encodes knitout instructions for faces. The only machine knitting constraints that are encoded in the definition is the requirement of maintaining the directed-acyclic property. This is strictly a machine knitting constraint, since e.g., a hand knitter can access any dropped loop, tie yarn ends together or carefully pull a part of the surface through gaps between or within loops! Beyond this, no machine specific information is encoded and the representation can be used for circular, flat or multi-bed machines.

An easy-to-edit representation The geometry and code face libraries make it easy to extend available library entries in a consistent manner. The geometry face provides a way to visualize stitches as spline paths, the same geometry face can be constructed by many different code faces. The mesh itself is similar to a polygon-soup mesh representation of the embedded 3D structure with explicit connections for tracking dependencies. In the next section I describe an intuitive system using a basic library of faces (illustrated in figure 5.4) that enables editing the augmented stitch mesh in the output 3D space while maintaining machine knittability.

5.2 *Editing an Augmented Stitch Mesh*

Any editing operation that takes as input a set of augmented stitch mesh faces and produces a modified set of augmented faces (i.e., with appropriate dependencies and face programs) without changing knit-

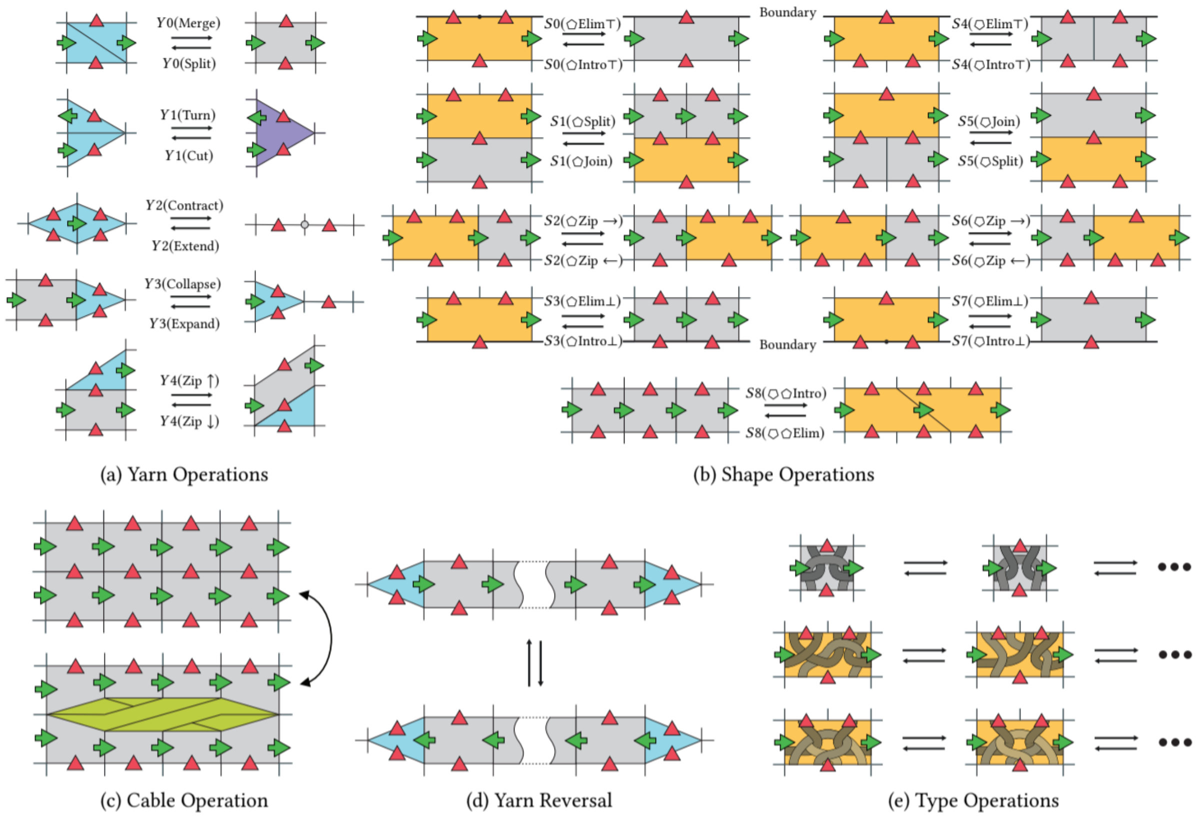


Figure 5.7: (a)Yarn can be manipulated with blue yarn-end face operations, turned with purple short-row faces; (b)orange pentagons introduce increase and decrease shaping; (c) Faces can be of complex types such as the green cable face; (d) Edge label directions can be reversed to change yarn direction; (e) Face programs can be edited to change type without changing edges. Green and red arrows indicate yarn direction and loop direction respectively.

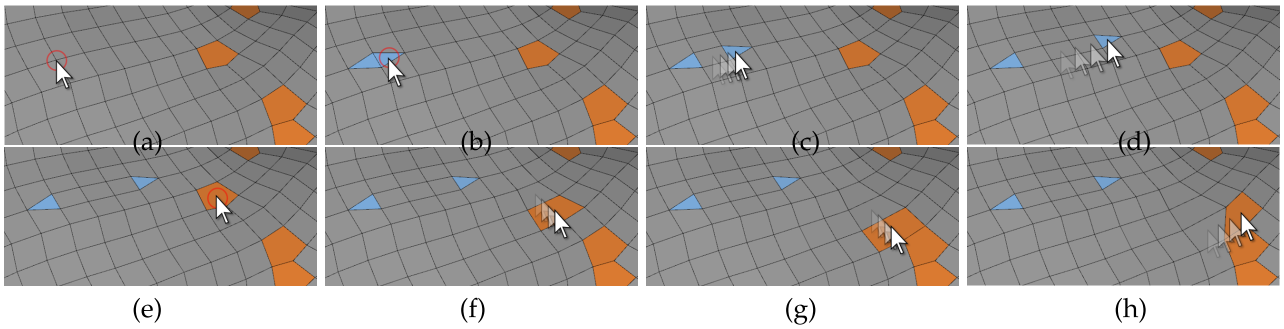


Figure 5.8: Applying *Y*2 in (a) and *Y*3 in (b - d) to introduce a small piece of yarn in a "zipper" like movement and aligning the shape pentagons, again using a "zipper" like motion (*S*5 in (e - g) and *S*6 in (h)).

tability constraints is *valid*. For an intuitive user interface experience, we specifically design a series of *shape* and *data* editing operations (Figure 5.7, 5.8). These editing operations are inspired by quad-mesh editing approaches explored in graphics research (Peng and Wonka [2013]). It is possible to construct a globally inconsistent structure with local editing operations violating 5.2. However, it is easy to perform a topological sort to detect such cases and restrict the system from performing these operations thereby maintaining validity. Although this might appear restrictive, the editing operations presented can effectively span the space of augmented stitch meshes within the same topology class (as defined by the underlying Reeb graph).

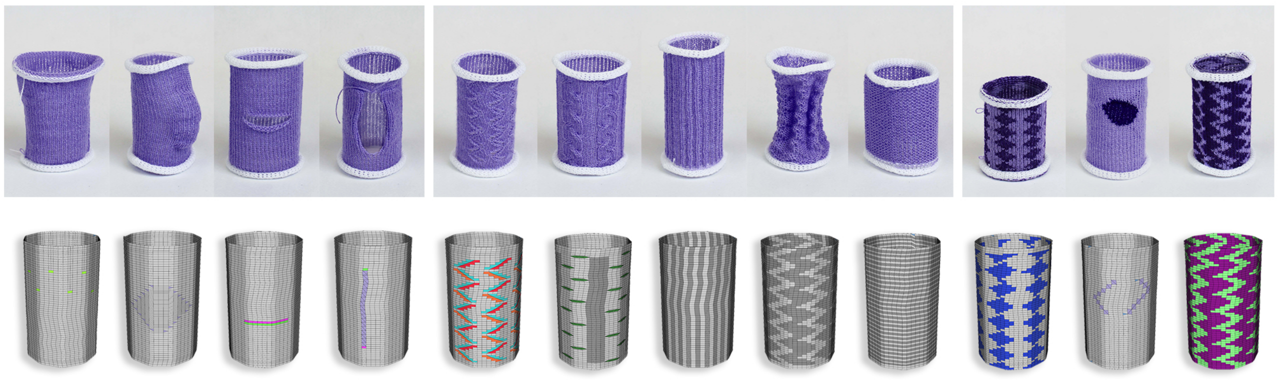


Figure 5.9: A sampling of various editing techniques; created by editing a cylindrical augmented stitch mesh.

The suite of editing operations provided allow users to navigate the space of knittable augmented stitch meshes (Figure 5.7). These operations all involve replacing some portion of an augmented stitch mesh while maintaining compatible edge labels. Edits of this sort can still introduce global dependency cycles as shown in figure 5.10, but they can be identified and restricted.

Editing operations can be split into mesh editing and data editing operations. Mesh editing operations change the mesh structure (face counts or connections) and include *yarn operations* that deal with yarn start/end faces, *shape operations* that modify pentagons (for increasing or decreasing loops), and *cable operations* that add or remove cable faces(for reordering loops after construction). Data editing operations do not change the mesh structure, and include *type operations* that change face type and *yarn reversal* that reverses yarn direction along a row. Although the stitch mesh structure might be modified by mesh editing, the genus of the input surface is not modified by any of the editing operations.

Yarn Operations involve manipulation of single-yarn-edge triangles (Figure 5.7a). Merging two yarn-start/end triangle faces over a *loop* edge will either form a regular quad, $Y0$(Merge), or short-row face to turn the yarn based on the types of remaining four edges, $Y1$(Turn). On the other hand, one row can be broken into two rows by their reverse operations. Removing or adding pair of yarn-start/end triangles can be used to add or remove a row, $Y2$(Contract/Extend). Yarn-start/end triangles are allowed to move along the loop-wise direction as well as along the yarn-wise direction ($Y3$ and $Y4$).

Shape Operations Shape operations allow the user to move pentagons along the loop-wise and yarn-wise direction as illustrated in Figure 5.7b ($S1$, $S2$, $S5$, and $S6$). Note that operations $S0$, $S3$, $S4$, and $S7$ can remove/add a vertex without causing problems because they do so at the boundary of the mesh.

Transposing loops after knitting them create interesting *cables*. These patterns can be supported by insertion and removal (Figure 5.7c) of cable faces of any length between two rows of regular stitches. These faces do not have yarn edges, so they cannot construct new loops, only rearrange them.

Type Operations These simple editing operations change face program associated with a face (as long as the two programs are compatible – have the same edge labels on the face). For example, a "knit" face program can be swapped for a "purl" face program. The system will use the corresponding face program to generate machine code during instruction generation.

Yarn Reversal In addition to these face modifications, our system also includes an operation for reversing yarn direction by changing the face types and internal edge labels of an entire row (Figure 5.7d). While not strictly necessary, this operation is much more convenient than removing and re-inserting a yarn stitch-by-stitch to change its direction.

These edits work together to enable natural dragging-based edits, where faces are moved across the mesh, locally altering topology. For example, Figure 5.8, users can "zipper" in and out partial rows of yarn by moving yarn start or end faces, and do the same with columns of loops by moving increase or decrease faces. This gives users access to both "short-row" and "increase-decrease" shaping techniques in a very intuitive way. This dragging-based interface is inspired by singularity editing interfaces such as the one proposed by Peng and Wonka [2013], but also preserve the machine knittability.

5.2.1 *Preserving Machine Knittability*

Though our local edits will never introduce locally conflicting edge labels, they are not always legal to apply because they can potentially introduce a dependency cycle as shown in in Figure 5.10. When editing, our interface checks the legality of each operation by attempting a topological sort on the dependency graph induced by the edge labels; if a directed cycle is found between a face and itself, then the DAG property has been violated and the edit is not permitted.

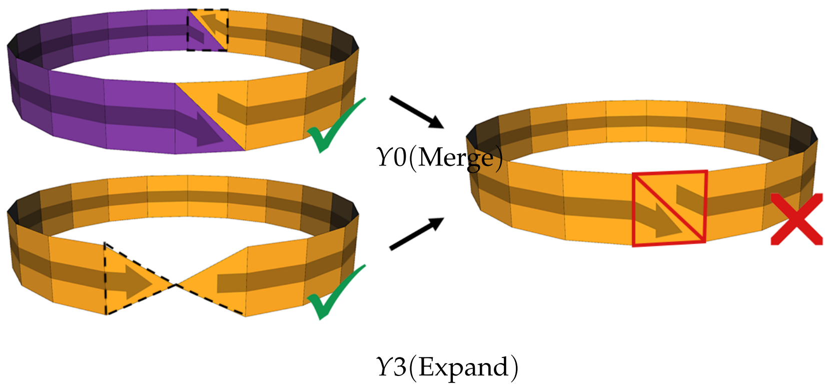


Figure 5.10: Examples of edits (shown with dashed edges) that the interface would prevent because the resulting mesh contains a cyclic dependency between faces. Arrows show yarnwise dependencies, and loopwise dependencies (not shown) point from bottom to top. Yarn-end and yarn-start faces (highlighted in red) form an unknittable structure by introducing a cyclic dependency.

Generality We refer to an editing operation that passes the global ordering check, and thus can be executed, as *valid*. Importantly, there is always a sequence of valid editing operations that can be used to transform one machine-knittable augmented stitch mesh into another (of the same input topology).

Indeed, we can prove a restricted version of this statement:

Proof. As shown in Figure 5.11, by repeatedly applying operation $S5$($\pentagon$Split), the decreasing face can be moved to the top boundary and the stitch mesh remains valid ($S5$ does not introduce local cycles). If the pentagon is trapped by a yarn-end face, the yarn-end can be moved (because the operations $Y1$ and $Y2$ do not introduce cycles), and the pentagon can be moved to the boundary. Then, operation $S4$($\pentagon$Elim$\top$)

can be used to remove a decrease pentagon. Similarly, repeatedly applying operation $S1$(⬠Split) and $S3$(⬠Elim⊥) can be used to eliminate increase pentagons. □

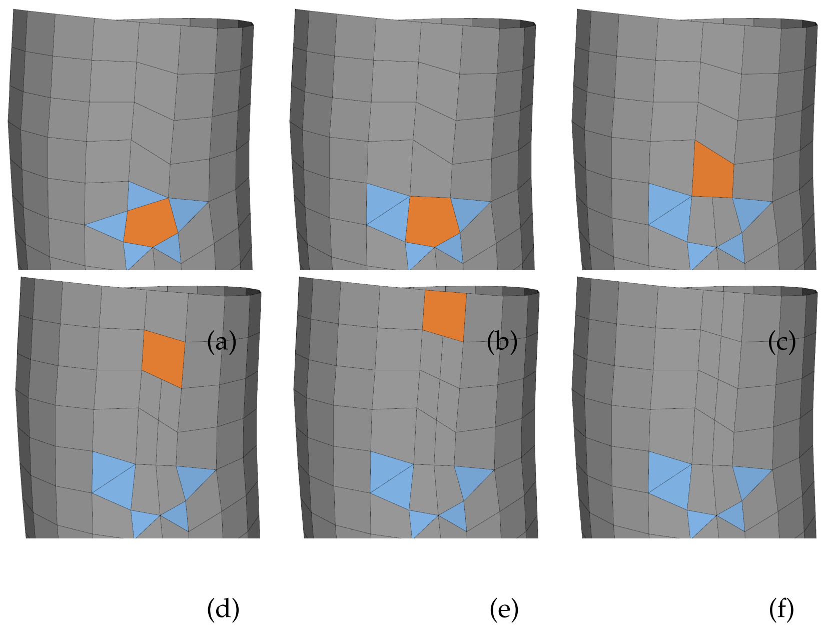


Figure 5.11: **Removing a pentagon while preserving knittability:** (a - b), move the yarn-end face above the pentagon; (c - e), move the pentagon to the boundary; (f), remove the pentagon.

Theorem 5.1:. The editing operations supplied by our interface are sufficient to connect the space of all machine-knittable augmented stitch mesh tubes.

Proof. First, we show that any valid stitch mesh tube can be turned into a trivial pattern. Any pentagon face can be edited out without breaking validity by repeatedly applying operation $S5$ and moving the face to the boundary. Operation $Y1$(Cut) can be used to remove any yarn-turn triangles. Finally, all yarn-start triangles can be moved closer to their yarn-end by repeatedly applying operation $Y2$(Contract) and $Y3$(Collapse). The result is a stitch mesh consisting only of a ring of edges and no faces.

Finally, these edges can be collapsed to a ring with just two edges by applying $Y2$(Extend) and $Y3$(Expand) to fill the ring with a single row of quads, followed by $S4$(⬠Intro⊥) and $S0$(⬠Elim⊤) to reduce the number of quads to one, followed by $Y2$(Contract) to remove the yarn.

Let $f_1, f_2, \cdots, f_n$ be the sequence of n operations to turn a stitch mesh F into the trivial pattern. Let $g_1, g_2, \cdots, g_m$ be the sequence of m operations to turn a stitch mesh G into the trivial pattern. If A is machine-knittiable, it passes the ordering check, since $g^{-1}(g(A)) = A$,

g^{-1} must be valid on $g(A)$. Hence, $g_1^{-1} \circ g_2^{-1} \circ \cdots \circ g_m^{-1}$ is valid on the trivial pattern, so $g_1^{-1} \circ \cdots \circ g_m^{-1} \circ f_n \circ \cdots f_1$ is valid on F and results in G. □

A similar, more general, proof can be conducted for non-tubelike meshes by using a variant of the same construction, but some care must be taken to keep the "trivial configuration" compatible with the underlying topology.

5.2.2 *Topological Edits to the Augmented Stitch Mesh*

The editing setup described above cannot change the topology of the shape. Next, I describe local operations that can be used to change the topology of the augmented stitch mesh:

1. Creation of new components. This is straightforward, by introducing the ability to create a yarn-in face, a knit face and a yarn-out face a new component can be introduced. This component can then be edited with the local edit operations described in 5.7.

2. Merging two components into one. We introduce a gluing operation that allows two loop edges to be connected to merge components. Yarn operations described above can be performed after merging to connect and minimize the number of yarns used in the augmented stitch mesh.

3. Splitting a component into two components. Two separate components can be glued together to incorporate splitting as well. After gluing, yarn operations can be used to reduce the number of yarns being used in the components.

4. Sheets and tubes. Operations $G0$ and $G1$ also allow users to connect edges of a sheet into tube. The edit operation for changing the yarn direction (Fig 5.7(d)) locally is useful to introduce the correct yarn order expected in a sheet and a tube.

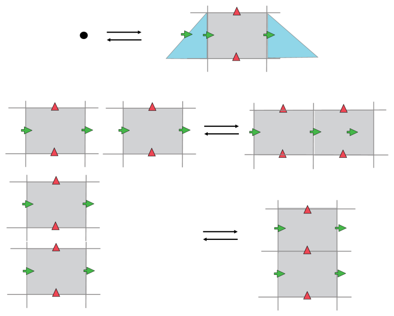


Figure 5.12: (top) C:A new component can be created (bottom left) $G0$: two yarn edges can be identified or connected (bottom right) $G1$: two loop edges can be identified or connected).

Similar to the previously described edit operations, any gluing that violates DAG properties is not permitted. During merging and splitting, balance of the augmented stitch mesh may be violated. This is not explicitly prevented or handled as there are multiple ways to deal with balance including introducing appropriate face types to fix balance by locally increasing slack or changing the number of loops produced or consumed. A production system can highlight slack imbalance during a merge or split operation to the user for immediate attention.

Additionally, since non-manifold structures can be supported, the system needs to introduce the ability to produce:

1. T-joins along the loop-direction.

   Supporting T-joins along loops can be introduced by adding a non-manifold variant of the decrease face and operation $S4$ – that takes two loops from potentially two different layers and move them to the same output location. Essentially, this allows for attaching a sheet component onto two sheets or a tube loop-wise creating a vertical join as illustrated by operation $T1$.

2. T-joins along the course direction.

   To support T-joins along the course direction, we introduce a similar variant of the turn face and operation $Y1$ that connects two yarn edges of the same course using a two faces as illustrated by operation $T2$ and extended by operation $T3$.

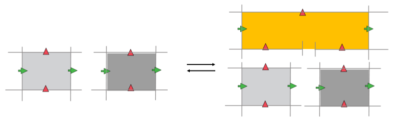


Figure 5.13: Introducing T-junctions along loop-directions with operations $T1$.

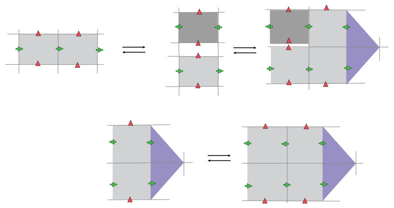


Figure 5.14: Introducing T-junctions along yarn-directions with operations (above)$T2$ and (below)$T3$.

The T-join operations can potentially *connect* edges on faces with opposing orientation. However, the overall embedding of the surface apart from these isolated connections needs to be non-intersecting for scheduling the mesh on the machine. The interface presented here does not enforce such an embedding automatically, but allows the user to edit the embedding of the augmented stitch mesh with selection and transformation tools. Even without an embedding check, this provides a useful starting point to visualize, simulate and render complex knitting patterns.

Supporting these operations enable a modelling interface that allows designers to construct knittable geometry from scratch in intuitive ways. Note that there may be multiple ways to achieve these surface modifications since many different face programs can be viewed as creating the same type of of surface. For maintaining the visual appearance of the stitch mesh, during editing, vertex positions can be updated. In our implementations, we use projective dynamics (Bouaziz et al. [2014]) to update the mesh geometry. From the *ShapeOp* library by Deuss et al. [2015], edge-strain constraints are used to ensure faces retain the approximately correct size and bending and plane constraints are used to maintain the 3D shape.

The augmented stitch mesh and supported editing operations provide an effective way to represent machine knittable structures. Next, I will describe how these augmented stitch meshes can be generated from 3D models.

6
Constructing Machine Knitting Patterns from 3D meshes

The augmented stitch mesh provides a suitable discrete representation for machine knitting patterns. With the editing operations expressed in the previous chapter, these meshes can be created from scratch. However, there are a large number of 3D models available in the wild and an automatic pipeline to turn these models into augmented stitch meshes without explicitly constructing them face by face, would be useful.

A class of techniques well studied in graphics and computational geometry involve cutting of polyhedra and parametrization of 3D surfaces. When constructing structures with flat materials such as flat fabric or paper, such techniques are clearly relevant. The surface is flattened by cutting, and flat materials can be glued together along the appropriate cuts to create a curved surface. Unfolding based techniques can be used for machine knitting as well. Because cutting and flattening produces a flat pattern – such augmented stitch meshes can be constructed with linear flat bed or circular machines. Apart from being amenable to simpler machines these patterns are likely to be faster and more robust than patterns with transfers.

Graphics research also abounds with techniques to remesh and convert surfaces with triangle or other polygonal representations into quad-dominant meshes (Bommes et al. [2013]). The 'knit face' in an augmented stitch mesh is in fact a quad with some constraints on how edges go together and with edge lengths prescribed by stitch dimensions. Given that a large fraction of a knit object consists of knit (front or back) faces, augmented stitch meshes are typically quad-dominant meshes. Any anisotropic quad-dominant mesh can be viewed as an augmented stitch mesh if it satisfies the following:

1. Edges can be labelled with directed labels such the mesh satisfies directed acyclic constraints.
2. A local face program can be written for each unique face (defined by its directed edge labels)

3. Each face matches the dimensions of the stitches it represents.
4. The mesh matches the shape of the underlying shape (under some suitable matching function).

In this chapter, I describe strategies for constructing augmented stitch meshes from manifold triangle meshes built on ideas of quad meshing and unfolding.

6.1 *General Patterns*

I present two strategies based on quad-meshing approaches to construct an augmented stitch mesh structure. The first is an incremental remeshing approach and the second is a faster hierarchical remeshing approach.

Using a time field as a guide Once an augmented mesh is created, its validity can be checked by assigning the time function implied by its yarn and loop-order and testing its Reeb graph's embedding based on hardware requirements. However, to construct an augmented stitch mesh it is often useful to guide the remeshing process with a time function. The user can directly influence the time function by specifying starting and ending cycles for knitting that are used as boundary constraints and interpolated smoothly within the surface as shown in figure 6.1.

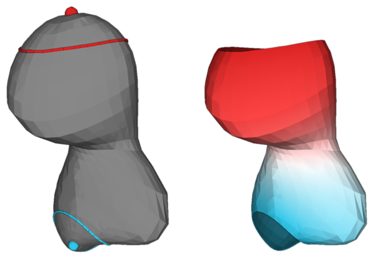


Figure 6.1: A simple interface that allows users to generate suitable time functions for arbitrary meshes by specifying a sparse set of constraints along mesh edges. Laplacian interpolation is used to extend these constraints to a time function over the mesh (represented by the color map, -1 🟦🟥 1, in the figures).

In the absence of a user preference, using the first eigen-function of the cotan-weighted mesh laplacian, also called the fiedler vector, usually provides a good order for the mesh faces (Levy [2006]).

Problem Statement 6.1: Augmented Stitch Mesh Generation. Given a triangulated, oriented, manifold input mesh $\mathcal{M}$ and a scalar function f that defines a knitting time on the surface of $\mathcal{M}$, we wish to generate an augmented stitch mesh representation.

6.1.1 *Incremental Remeshing*

The incremental approach produces an augmented stitch mesh that approximates the input mesh in three main steps. First, *remeshing* creates a specially-structured knitting graph on the surface. Second, *tracing* connects the graph into a single continuous yarn path from which directions for operations (stitch faces) can be identified. Finally, the dual of the traced knit graph is constructed with appropriate basic face types to generate an augmented stitch mesh.

Knit objects have an intrinsic row-column structure, where the rows arise from yarn-wise connections, and the columns arise from loop-wise connections. (See Figure 6.3).

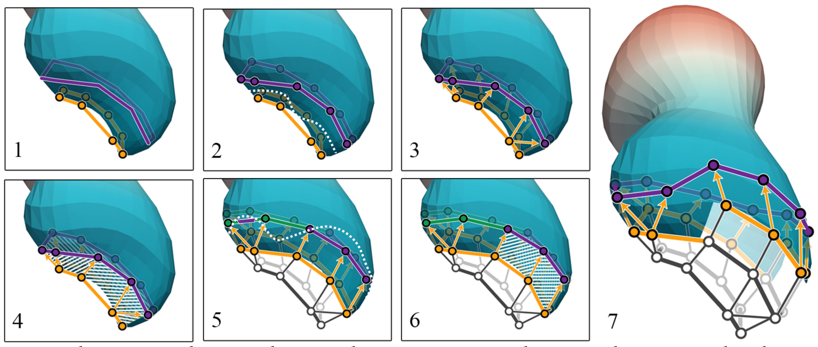


Figure 6.2: During remeshing, a graph with row and column edges is constructed iteratively by: (1) finding a next cycle (purple) one row-height away from the current cycle (yellow), (2) adding nodes to the next cycle and marking nodes to keep (purple) or discard (green) based on the time function, (3) linking the nodes to the active cycle, (4) discarding the marked nodes and trimming the mesh, and (5-7) updating the active cycle and repeating. The level set of t_{active} is shown as a dotted white line. If the next cycle occurs after t_{active}, it is entirely accepted (2-3) otherwise segments that lie under the contour are accepted if longer than $2l_r$ (5-6).

In the remeshing phase, the incremental remeshing method produces a directed graph to guide the row-column structure of the final knit object. This graph needs to be suitable for tracing, follow the input knitting time function, and approximate the input surface.

That is, remeshing creates a knit graph $(\mathcal{N}, \mathcal{R}, \mathcal{C})$:

$\mathcal{N} \equiv \{n, \ldots\}$	○	nodes
$\mathcal{R} \equiv \{(n_i, n_j), \ldots\}$	⋯	directed row [yarn] edges
$\mathcal{C} \equiv \{(n_i, n_j), \ldots\}$	↑	directed column [loop] edges

Each node of the remeshed graph represents two knit loops in the fabricated pattern, which is useful during tracing.

There are a few useful properties that make a directed graph knittable. To be suitable for tracing, the graph should have consistent orientation and be helix-free (Properties 6.1, 6.2). Further, to satisfy slack constraints (Property 4.1) and limit the number of basic face types,the graph must satisfy a limited node degree and it must allow a valid balanced layout (Properties 6.3, 6.6). Finally, the graph needs to follow the time function and approximate the input geometry (Properties 6.7, 6.8). Properties 6.4 and 6.5 simplify the graph structure at short-rows and when cycles undergo a change in topology; although our knit graphs satisfy Property 6.4 and 6.5 by construction, they are not necessary conditions for knittability.

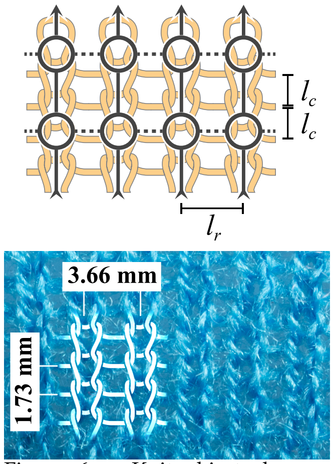
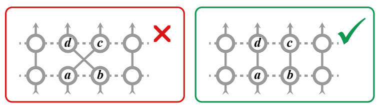


Figure 6.3: Knit objects have a row-column structure. In our system, this structure is captured by the knit graph, whose nodes represent two stacked stitches (i.e., each row represents two physical *courses* or knit rows), and whose edges are constrained to be close to measured stitch dimensions (l_r, $2l_c$).

Property 6.1: Consistently Oriented. Adjacent rows should be consistently oriented. That is, for all nodes a, b, c, d:

$$(a, b) \in \mathcal{R} \land (a, c) \in \mathcal{C} \land (b, d) \in \mathcal{C} \implies (d, c) \notin \mathcal{R}$$

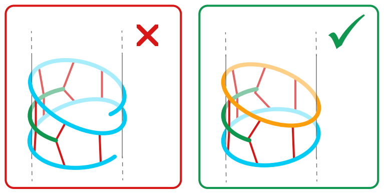


This reflects the fact that cycles on the machine bed cannot be reversed.

To maintain consistency with the orientation of a surface (that this graph approximates), a stronger constraint of handedness must be employed – edges around a node cycles through incoming column edges, outgoing yarn edges, outgoing column edges and incoming yarn edges[1]. However our tracing strategy (described in 6.1.2) of doubling nodes and row edges effectively treats all nodes as consistently handed.

[1] Thanks to Ben Jones for suggesting this stronger property.

Property 6.2: Helix-Free. The column edges should form a partial order on the rows. That is, treating row edges as undirected, there should be no paths from the end of a column edge to its start.

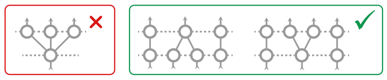


$$\forall (a,b) \in \mathcal{C}^+ : (b,a) \notin (\mathcal{R} \cup \text{reverse}(\mathcal{R}) \cup \mathcal{C})^+$$

where $^+$ denotes transitive closure, i.e., paths in the graph.

Arbitrary helices break the row structure of knitting and cannot be traced in general although the final pattern generated for knitting a tube is, in fact, helical.

Property 6.3: Limited Node Degree. Each node n must correspond to a constructable type of stitch.

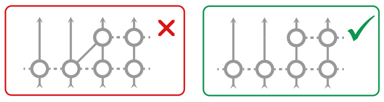


That is, it should have at most two row edges (one in, one out) and at most two in and two out column edges.

$$\forall n, |\{(x,n) \in \mathcal{R}\}| \leq 1, |\{(n,x) \in \mathcal{R}\}| \leq 1,$$
$$|\{(n,x) \in \mathcal{C}\}| \leq 2, |\{(x,n) \in \mathcal{C}\}| \leq 2$$

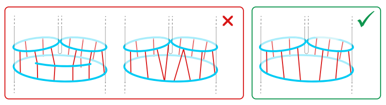
The constraint on row (yarn) edges is tight – stitches are always formed from one piece of yarn – while the constraint on column (loop) edges reflects the capabilities of the machine being used. Particularly, handling multiple incoming column edges requires the machine to stack multiple loops on one needle, so is limited by the size of the needle's hook. Handling multiple outgoing column edges requires either splitting a loop, which can stress yarn, or casting-on additional stitches, which can create a small gap in the fabric. In both cases, we have chosen the conservative limit of two edges, though nothing in our pipeline intrinsically depends on these limits.

Property 6.4: Simple Short-rows. Each terminal end of a short-row must have a single in and out column edge.



$$\forall n, |\{(x,n) \in \mathcal{R}\}| = 0 \Rightarrow |\{(x,n) \in \mathcal{C}\}| = 1 \wedge |\{(n,x) \in \mathcal{C}\}| = 1$$
$$\forall n, |\{(n,x) \in \mathcal{R}\}| = 0 \Rightarrow |\{(x,n) \in \mathcal{C}\}| = 1 \wedge |\{(n,x) \in \mathcal{C}\}| = 1$$

Property 6.5: Simple splits and merges. At merges and splits, rows are complete cycle and column edges incident on its nodes are linked 1-1.



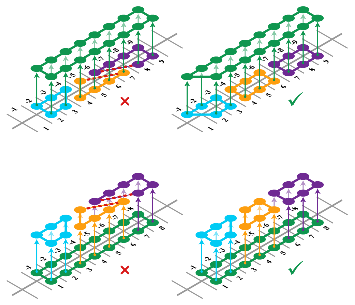
$$\forall (m,n),(m',n') \in \mathcal{C} \text{ s.t. } (m,m') \in \mathcal{R}^+, (n,n') \notin \mathcal{R}^+ :$$
$$|\{(x,n) \in \mathcal{C}\}| = |\{(m,x) \in \mathcal{C}\}| = 1,$$
$$|\{(m,x) \in \mathcal{R}\}| = |\{x,m) \in \mathcal{R}\}| = 1,$$
$$|\{(n,x) \in \mathcal{R}\}| = |\{(x,n) \in \mathcal{R}\}| = 1$$

where, $\mathcal{R}^+$ indicates transitive closure, i.e., paths along row edges.

This property ensures that each generalized cylinder starts and ends in a cycle (and not a short-row) and these cycles line up without requiring additional shaping operations. This property simplifies graph generation for tracing.

Properties 6.4 and 6.5 simplify linking and tracing. The constraints imposed by them additionally imply that increase and decrease shaping only occur between row-wise connected nodes.

Property 6.6: Feasible splits and merges. At splits and merges, nodes of both participating rows must have a feasible layout on the machine. Otherwise, splits and merges can cause yarn stress – red dashed lines show yarn stretching beyond stitch width, which cannot be laid out on the machine as shown by the illustration on the right.



Let $\bar{\mathcal{C}}$, $\bar{\mathcal{R}}$, $\bar{\mathcal{N}}$ be the column-edges, row-edges and nodes restricted to these rows respectively. A layout function l over $\bar{\mathcal{C}}$, and its extension l_n over $\bar{\mathcal{N}}$, must exist such that rows in $\bar{\mathcal{R}}$ are *not stretched* for all participating cycles:

$$\begin{aligned}
&\exists \quad l : \bar{\mathcal{C}} \to \mathbb{Z}, \quad l_n : \bar{\mathcal{N}} \to \mathbb{Z} \quad s.t. : \\
&\forall (a,b),(a',b') \in \bar{\mathcal{C}} : (a,b) \neq (a',b') \Rightarrow l(a,b) \neq l(a',b') \\
&l_n(a) \equiv l_n(b) \equiv l(a,b) \\
&\forall (a,b) \in \bar{\mathcal{R}} : |\ |l_n(a)| - |l_n(b)|\ | \leq 1
\end{aligned}$$

By Property 6.5, the function l can be consistently extended as l_n from $\bar{\mathcal{C}}$ to $\bar{\mathcal{N}}$.

Given a function l_n, a function l'_n can be constructed such that $\{l'_n(p_1)..., l'_n(p_n)\}$ is monotonic for every participating cycle $\{p_1, ..., p_n\}$ from some starting node p_1. Now, $\text{sgn}(l'_n(p))$ can be viewed as a needle bed and $|l'_n(p)|$ as a needle location. If the row edges are between adjacent needles (or across the bed), the yarn will not be stretched beyond the width of the stitch.

Property 6.7: Time-Aligned. The graph should respect the time field. That is, the column edges should increase in time and the row edges should remain about the same time:

$$\begin{aligned}
\forall (a,b) \in \mathcal{C} &: \text{time}(a) < \text{time}(b) \\
\forall (a,b) \in \mathcal{R} &: \text{time}(a) \approx \text{time}(b)
\end{aligned}$$

Rows can be monotonically ordered with respect to the given time function and when embedded on the mesh they follow the user defined time function i.e., row edges approximate level sets of the time function (Figure 6.4).

Property 6.8: Low Stretch. When the nodes are embedded in the surface, row and column edges should have lengths close to their corresponding knit features (Figure 6.3):

$$\forall(a,b) \in \mathcal{R} : \text{len}(a,b) \approx l_r$$
$$\forall(a,b) \in \mathcal{C} : \text{len}(a,b) \approx 2l_c$$

where l_r is the measured width of a stitch, l_c is the measured height of a stitch, and len() measures distance along the surface.

Construction In order to generate a graph with these properties, the algorithm proceeds by iteratively *slicing* the surface at a uniform distance from the boundary. Nodes are sampled on the slice and *linked* to previously generated rows. A portion of the linked region is then *trimmed* off the mesh, based on the time constraints. This process is repeated to generate a row-column graph over the entire mesh. An illustration is provided in Figure 6.2.

Recall in chapter 4, we described a slicing time function that may not slice a surface into an equal width band, the slice and trim approach ensures that equally spaced sub-slices are generated which can be executed on the machine with fixed size stitches and short-rows.

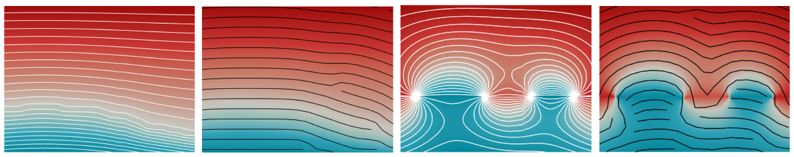


Figure 6.4: The rows generated by our remeshing algorithm **(black)** align well with the contours of the time function **(white)**.

Initialization To start, all mesh boundaries that contain a local minima of the time function are added to the *active cycle* set. Nodes are sampled along these boundaries with even spacing, and adjacent nodes are connected with row edges. The number of nodes is chosen to ensure approximately l_r units between adjacent nodes, and the normals of the mesh are used to consistently orient the row edges with the surface to the left.

Slicing Given a set of active cycles on the surface of the mesh, slicing generates a set of *next cycles*. Our code computes an approximate geodesic distance function on the surface of the mesh using the method proposed by Crane et al. [2013], starting at the active cycles, and takes the $2l_c$ level set of this function as the next cycles. The time function is linearly interpolated to all the vertices on the level set.

The next cycles so generated may not follow the time function. In order to guide the rows, our code trims the next cycles based on their time values. Let t_{active} be the maximum time encountered along the active cycle. If the next cycle entirely appears after t_{active}, then it is accepted; otherwise, all portions of the next cycles with $t > t_{\text{active}}$ are marked for discard and the remainder ($t \leq t_{\text{active}}$) are marked for acceptance (see Figure 6.2, panels 5 and 6). If the next cycle is not entirely accepted, the accepted segments form "short-rows" i.e., partial, non-cyclic rows. If the length of such a segment is less than twice the stitch width, $2l_r$, it is discarded, to avoid very small short-rows. If the first pass marked everything for discard, everything is re-marked for acceptance, i.e., if the entire cycle appears after t_{active} or if all segments that could have been accepted do not satisfy the minimum width $2l_r$.

Discarding segments based on time has the effect of keeping row edges approximately in line with the time function (Property 6.7), as demonstrated in Figure 6.4. Without this step, the generated patterns have no short-rows and can diverge from the time function, introducing arbitrary bind-off rows or rows that change shape rapidly (Figure 6.5).

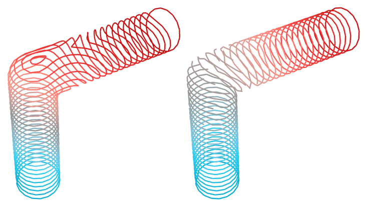


Figure 6.5: Using geodesic distances only produces layouts of stitches with no short-rows **(left)**. The time function guides the knitting by introducing short-rows **(right)**.

Of course, if a pattern without short-rows is desired, that can still be achieved by setting the time function to the geodesic distance from the starting boundaries.

When the level set intersects a boundary, the entire boundary cycle is accepted as the next cycle. This guarantees that the next cycles are indeed cyclic and not chains, reducing the number of special cases required in our code. This can increase distortion of the shape close to the boundaries. In a production system, boundaries could be handled separately.

Linking Linking is performed in two phases. First, *alignment pairs* are computed between active cycles and next cycles. The next cycles are adaptively sampled to generate nodes. Second, column edges are generated between the active and next cycle segments for each alignment pair.

Generating alignment pairs Every node on the active cycle is assigned a target next cycle that is closest to it. Similarly, for every vertex (of the mesh) on the next cycle, a target active cycle is assigned that is closest to it. Segments of the active cycle and next cycle are paired for alignment if their targets match mutually (Figure 6.6). If a next cycle is not chosen by any active nodes or vice versa, the cycle is not considered for alignment in that iteration and is recomputed for subsequent iterations.

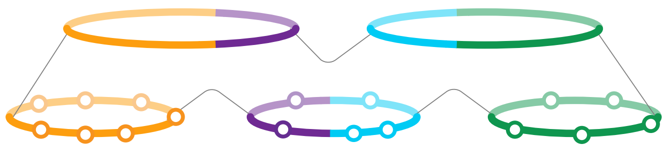


Figure 6.6: Segments of the active cycle and next cycle are paired for alignment if their targets i.e., closest cycle, match mutually

Sampling Alignment pairs allow adaptive sampling of the next cycles. The number of nodes required on the next segment is computed according to the stitch width, then clamped to maintain degree constraints (Properties 6.3, 6.4, 6.5). Stitch width is varied for each alignment pair to generate a valid number of next nodes.

Handling topology change In some cases, an active cycle can link to multiple next cycles (or vice versa). We refer to this as a split (or merge). In this case, all nodes are marked accept to maintain Property 6.4. Further, care must be taken during linking to ensure that the final structure has a feasible layout on the knitting machine bed (Property 6.6). If splits and merges are strictly binary, cycles can be rotated on the bed to ensure balance during layout. In the case of ternary or higher splits and merges, our code explicitly balances the layout locally.

Linking nodes Row edges are introduced between nodes on the next cycle. They are consistently oriented using surface normals. For each alignment pair, contiguous segments of nodes from the active cycle and the next cycle are linked to form column edges. As active and next cycles were extracted $2l_c$ distance apart from each other, our code now generates column edges that maintain this distance as closely as possible i.e., linking matches closest nodes by adding column edges subject to ordering (Property 6.1) and degree constraints (Properties 6.3, 6.4, 6.5).

Nodes in alignment pairs are linked to minimize cost. The cost of linking an active and next node is the squared distance between them on the surface of the mesh. The cost of linking an alignment pair is the sum of the costs of its constituent links. For each alignment pair with nodes $\mathcal{A}$ in the active segment and nodes $\mathcal{N}$ the next segment with links $\mathcal{L}$ between them:

$$\begin{aligned} \text{cost}(\mathcal{A},\mathcal{N}) \equiv \quad & \textstyle\sum_{(a,n)\in\mathcal{L}} \text{cost}(a,n) \\ a \in \mathcal{A}, n \in \mathcal{N}: \quad & \text{cost}(a,n) \equiv \text{len}^2(a,n) \end{aligned}$$

Our code determines optimal links using a dynamic-time-warping-like algorithm (Berndt and Clifford [1994]) that considers all valid combinations of 1-1, 2-1, and 1-2 links.

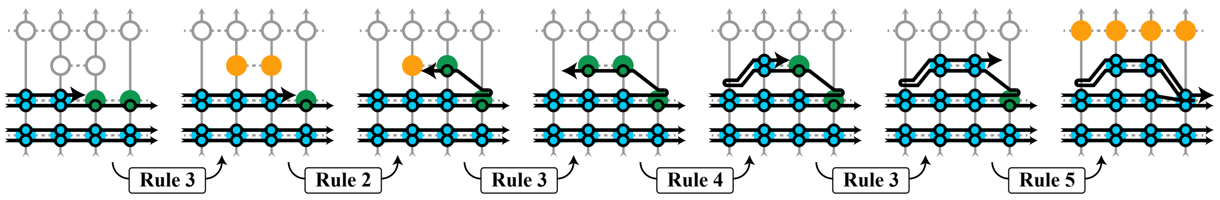


Figure 6.7: Tracing a short row on a small portion of a tube, according to the tracing rules. The rule applied between two steps is shown in the box below.

Trimming After generating links between the active cycles and the next cycles, the nodes on the next cycle marked for discard (along with any incident edges between them) are discarded. The remaining nodes and edges are added to the knit graph. Next, the portion of the mesh lying between the current active cycles and the *accepted* portion of the next cycles is removed. This trimming is accomplished by splitting the mesh along embedded edges of the knit graph and removing the faces bounded by row edges on the active and next cycle and column edges between them. Finally, the active cycle set is updated by reading off the nodes along the new mesh boundaries. Note that nodes on the boundary with outbound column edges are *not* considered part of the active cycles, as they have already been linked (panels 6 and 7 in Figure 6.10).

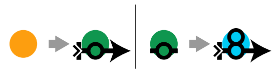
6.1.2 *Tracing the knit graph*

Tracing builds knitting instructions from the knit graph. Specifically, it traverses each node twice and makes two knit stitches at every node, connects nodes with yarn along row edges, connects nodes with loops along column edges, and traces the graph in the order defined by the edges.

The location of the current yarn is shown with an arrow (➤) designating its direction. The beginning (✕━) and the end (━✕) of a yarn are marked for clarity. Ends of short-rows are anchored by tucks (◒) where ◕ indicates *any* underlying node. In the figures below, column edges (↑) run bottom-to-top and row edges (┈) run left-to-right.

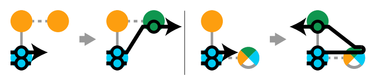
Tracing generates a list of knitting operations by traversing the knit graph using a set of local rules. These rules define an action based on the local context of the last last knitting operation performed with the current yarn. As it traverses, it will mark each node as knit once (●) or knit twice (8), and will query whether a node is ready (●) or not ready (○). Nodes are ready (●) if all column-wise predecessors of nodes in their row have been knit twice i.e., 8. In the following figures, the state of the knit graph before applying the tracing rule is shown on the left for each figure, and after applying the rule is shown on the right.

Tracing Rule 6.1:. Start yarn

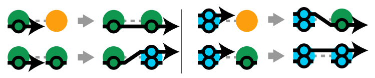


If there is no current yarn, start a new yarn by knitting an arbitrary ready or once knit node.

Tracing Rule 6.2:. Move to next row

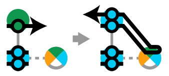


If the previous stitch made with the current yarn was at a node with a column edge to a ready node, then knit that ready node's row-wise neighbor (left). If this neighbor does not exist, tuck on the current stitch's neighbor and knit the ready node in reverse (right).



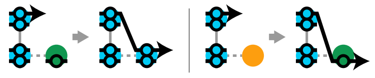
Tracing Rule 6.3:. Continue

If there is no column edge from the current node to a ready stitch, and the next stitch along the current row has not been knit twice, knit it.



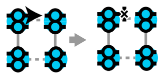
Tracing Rule 6.4:. Tuck and turn

Upon reaching the end of a short row, if the current node has not been knit twice, tuck at the current node's parent's row-wise next node, then knit the current node in the opposite direction.



Tracing Rule 6.5:. End short row

Upon reaching the end of a short row, if the current node has already been knit twice – knit the next stitch off the end of the short row continuing in the same direction.



Tracing Rule 6.6:. End yarn

If the current yarn can not be extended (i.e., no row-wise or column-wise adjacent ready node exists, and all adjacent nodes have been knit twice already), end it.

Together, these local rules cause tracing to walk along every row of the graph, knitting at every stitch, and tucking at the ends of short-rows. Figure 6.7 shows an example that applies the tracing rules on a small segment of a tube. Cases with row edges running right-to-left, and cases with node in/out degree two are handled similarly, and are not shown. Tracing will always succeed, by construction. The rows of the graph are always singly-linked chains or cycles (Property 6.3) and the graph is helix-free (Property 6.2). Hence, it must consist of cycle-shaped rows, possibly with intermediate non-cyclic short-rows. When a short-row becomes ready, by rule 6.2 tracing immediately proceeds to knit it. By rules 6.2, 6.4, 6.5, short-rows are reversed and traced twice, ending at the same stitch as it started and using the same yarn (see Figure 6.8). Thus, short-rows can be knit along with the cycle that precedes it – this cycle is unique because by Property 6.5, no short-rows appear at a merge. Ignoring short-rows, each generalized cylinder is a stack of cycles that can be traced by single yarn. When a cycle becomes ready, the previous cycle must have been completed and the same yarn is continued over using rule 6.2. For a cycle that lies at a split, it is possible that one of the split next cycles becomes ready before the

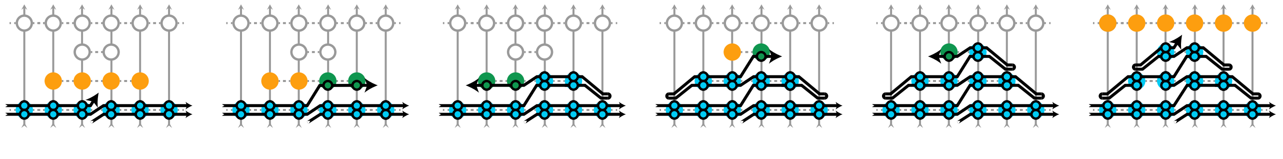


Figure 6.8: Tracing of short-rows starts and ends on the same stitch and can be viewed a part of the preceding full cycle.

previous cycle is finished. In this case, tracing proceeds to knit it and a new yarn is brought in to finish the cycle and continue on each of the remaining splits. When cycles merge, the yarn from the most recently completed cycle is extended. Thus, tracing uses at most $n + (k-1)m$ yarns to trace a knit graph with n starting cycles and m k-way splits (Figure 6.9).

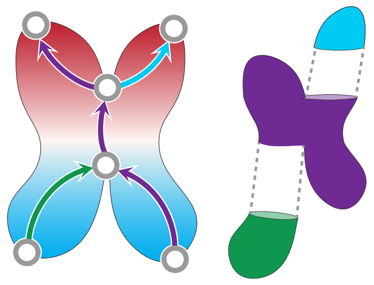


Figure 6.9: Tracing generates a yarn path for each cylinder representing an arc of the Reeb graph. For each initial active cycle, a yarn is introduced (purple and green). At the merge, the existing purple yarn from the most recently completed cycle is continued. At the split, the current purple yarn continues along one of the cycles and a new blue yarn is introduced for the additional cycle.

6.1.3 *Hierarchical Remeshing*

In contrast to the incremental approach, another approach is to quickly come up with a base mesh with approximately correct stitch heights based on contours. Then in subsequent steps, regions with errors can be updated and remeshed.

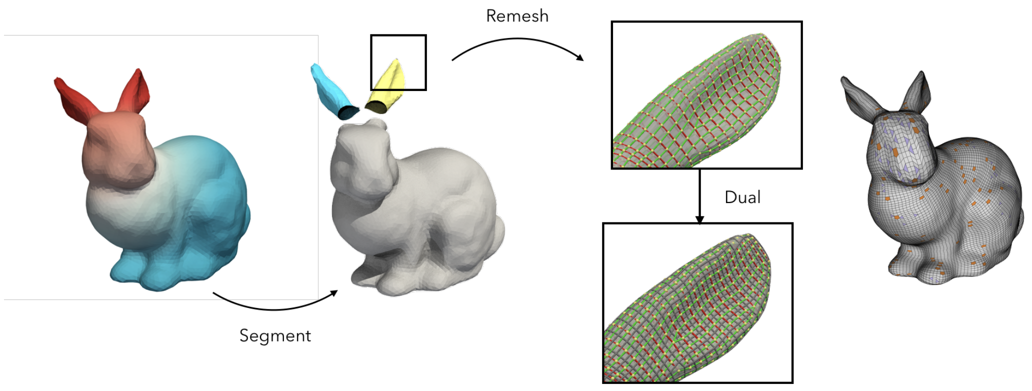
As a first step, the mesh is segmented into tubular regions using the input time function. The user may also edit boundaries for better alignment (as in Igarashi et al. [2008a]). Then, boundaries are discretized based on stitch width such that segments align one-to-one. Once the starting and ending boundary counts are computed, each segment is quad meshed based on the stitch dimensions l_r and l_c following Dong et al. [2005].

To maintain boundary lengths while limiting the change in stitch counts between rows for reliable fabrication, stitch counts are optimized by relaxing integer constraints on the counts and rounding the results:

$$s_f =_x \sum_i^n (x_i - s_i)^2$$
$$\text{subject to: } \frac{2}{3}x_{i-1} \le x_i \le \frac{3}{2}x_{i-1}$$
$$x_1 = s_1, x_n = s_n$$

where s_f is the vector of final stitch counts and s_i is the vector of initial counts.

Figure 6.10: The input mesh is first segmented into tubular segments using the input time function. Each tube-like region is rapidly remeshed into knit graph and aligned and connected along the boundaries. The traced knit graph is finally converted into an augmented stitch mesh.



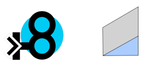
This quad mesh is interpreted as a knit graph similar to the one generated by the incremental approach. Nodes are placed at intersections, edges along the gradient of the input field are column edges and along the contours are row edges.

Refinement Column edges that are longer than $1.5l_c$ and row edges that are longer $1.5l_r$ are subdivided and new nodes are placed. A row edge is introduced between adjacent new nodes between column edges. Similarly, column edges that are shorter than $0.75l_c$ are removed and row edges that are shorter than $0.76l_r$ are merged. Any operation that violates the degree property 6.3 are avoided. Once no additional refinements can be introduced, the graph is traced as described in the incremental approach.

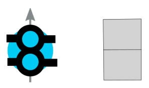
6.1.4 *Assigning stitch faces*

To convert this traced graph representation into an augmented stitch mesh, each traced node is converted into augmented stitch faces based on the edges incident on it:

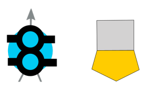
Start yarn A traced node with a starting yarn is turned into a *yarn-in* triangle face and a knit face.



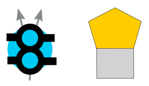
Regular node A traced with exactly with one incoming column edge and one outgoing column edge is turned into a stacked pair of *knit* quad faces.



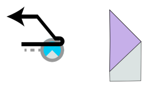
Decrease node A traced with two incoming column edge and one outgoing column edge is turned into a *decrease* pentagon face and a stacked *knit* quad faces.



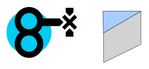
Increase node A traced with one incoming column edge and two outgoing column edge is turned into a *knit* quad face and an *increase* pentagon face stacked on it.



Turn node A tuck and turn node is turned into *turn* quad face stacked over the previous node face.



End yarn A traced node with an ending yarn is turned into a knit quad face and a stacked *yarn-out* triangle face.



In this dual representation, faces are connected along yarn-wise edges if the nodes are connected by a row edge. Similarly, along loop-wise edges when the nodes are connected by a column edge.

6.1.5 *Meshing quality*

To compare the stitch sizes of the resulting augmented stitch obtained from the two remeshing strategies, we plot the relative error in the length of each edge from its expected length l_r and l_c. Our remesh-

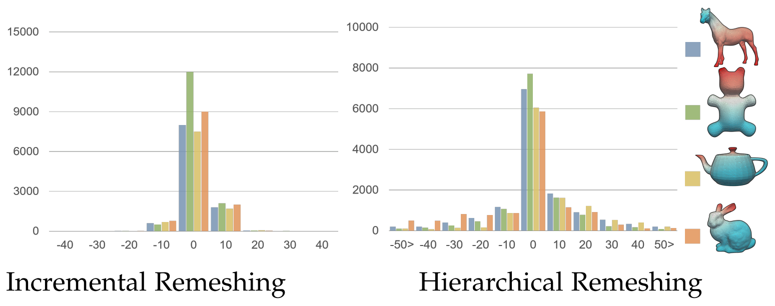


Figure 6.11: (Left) Most edges have less than 10% length error introduced due to fabrication constraints and discretization. (Right) Most edges have less than 20% length error

ing step seeks to produce a low-stretch knit graph (Property 6.8). In general, it succeeds, with the majority of edges within 10% of their target length (Figure 6.11) for the incremental approach and 20% of their target length for the hierarchical approach.

As expected, the incremental approach is slower but has fewer errors than the hierarchical approach. The bunny model takes 11 mins to remesh using the incremental approach and under 30 seconds with the hierarchical approach on a 2.7GHz Intel Core i5 Macbook Pro with 16GB RAM.

The geometric accuracy of the meshing is limited by the size of the stitches used to knit them. This size, in turn, depends on the gauge of the machine and is typically in the order of millimeters. Features smaller than the stitch size cannot be represented. Knitting machines can change row lengths using shaping operations at a limited rate (Property 6.3). This limits the amount by which the radius of a generalized cylinder can be varied along its rows and thus affects the approximation of the mesh by the knit graph.

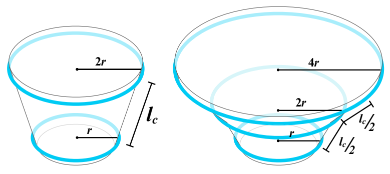


Figure 6.12: Decreasing the stitch size uniformly increases the number of stitches in each cycle and thus increases the maximum width achievable by the next cycle.

However, under refinement of stitch size, the accuracy of our remeshing improves. To see why, consider a cycle of a cone at radius r with $n = \frac{2\pi r}{l_r}$ stitches on its circumference. Due to property 6.3, the number of stitches in the next cycle at a distance l_c apart, is at most $2n$ and its radius can be at most $2r$. If the size of the stitch width and height is reduced by a factor of 2, the number of stitches in a cycle of radius r is now $2n$ allowing a wider radius of $4r$ at a distance l_c on the surface – in the limit, as the stitch size reduces, any surface feature can be accurately represented.

For a reasonable scale and stitch size, the incremental remeshing approach accurately captures bending angles using short-rows as shown by these tubes with progressively increasing bend angles in figure 6.13.

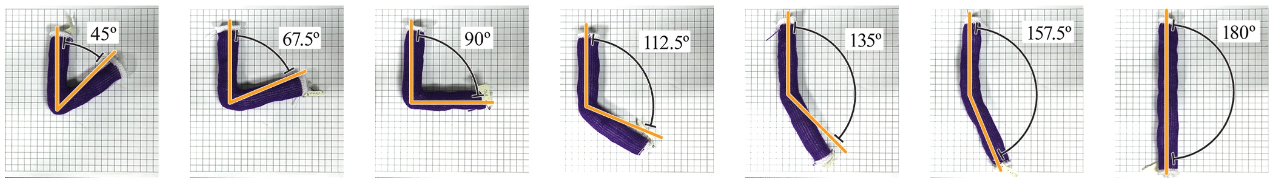


Figure 6.13: A series of bent tubes fabricated using the incremental remeshing algorithm. Short-rows are automatically produced by the procedure and accurately generate the bending angle.

6.2 *Transfer-Free Patterns*

The approaches described so far slices a surface and interprets those slices as rows of knitting, beginning from the boundary or some user-specified region. Since each row can be of a different length, in general, *increases* and *decreases* (or pentagon stitch faces) must appear in the discretization. These row length changes are executed using the `xfer` instruction that *moves* loops around and use both the beds of a two-bed machine to do so. Single-bed consumer knitting machines and even circular industrial machines cannot perform any transfers at all. Further, since transfer operations execute in a different pass on current machines, patterns that need transfers tend to take longer to fabricate. Transfer-free or short-row-only patterns therefore form an interesting subset of sheet-like patterns.

A pattern that entirely consists of only short-rows can be implemented without any transfer operations. Because short-rows knits over some portions of the surface selectively, they can also be viewed as introducing cuts in a pattern as well. The cuts are glued (by knitting) as a part of the construction process. This idea can be formalized such that short-row-only patterns can be viewed as a subset of flat cut-and-sew-like patterns. Figure 6.14 illustrates this relationship using paper craft. A sheet of paper is flat if for any point[2] on it, the total angle around it is $\theta = 2\pi$. The curvature K around a point can be measured as the angle deficit needed to make the point flat: $K = 2\pi - \theta$. To create positively curved cone shape (see figure 6.14 top), a wedge of material can be removed and the edges can be glued together. To create a negatively curved shape (figure 6.14 bottom), excess material is glued into the cut. Note that to achieve a cone angle at a point, the direction of the cut does not matter. Or alternately, to flatten the shape at a cone angle, it can be cut along any seam terminating at that point.[3] However, to glue the cut by knitting, identified points should line up along the knitting direction. During fabrication, any point on the surface is constructed as a part of some row. The distance between any point on the surface and any path parallel to the construction direction in the flattened pattern must be equal to the shortest distance between them on the surface because the rows are orthogonal to the knitting direction by definition. Also, since each row is orthogonal to

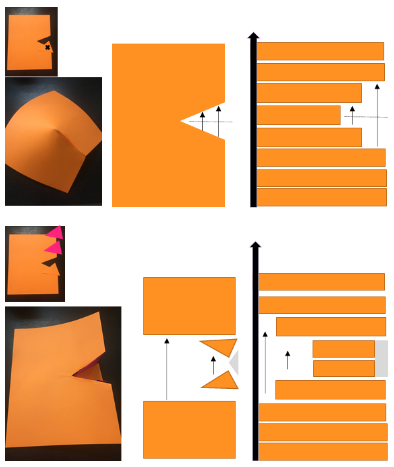


Figure 6.14: Short-rows can be interpreted as a flat pattern with cuts where the angle bisector of the cut is orthogonal to the construction direction.

[2] On a triangle mesh, the discrete (Gaussian) curvature around any vertex, the total angle θ can be measured by summing up the corner angles of each triangle incident on the vertex. At every other point, the mesh is flat.

[3] In cut-and-sew patterns, *darts* can be rotated for the same reason.

the knitting direction, for the cut to align, its angle bisector must also be orthogonal to the knitting direction.

In the illustration, the knitting direction is shown by the black arrow and the rows of the pattern are shown by orange strips. Given a knitting direction, there are only two (symmetric) ways to place the cut orthogonal to the knitting direction and the orientation of the surface can be used to pick a canonical direction. For cuts around negatively curved points, this argument might seem a little problematic because the excess material creates an overlap. However, the flat pattern can be entirely cut at the negatively curved saddle vertex and separated in time. As long as the cut is appropriately aligned, the knitting process continues to produce a connected surface.

Concretely, a flat pattern with cuts (identified for gluing) can be interpreted as a short-row-only knitting pattern only if the angle bisector of each cut orthogonal to the knitting direction. Equivalently, the distance between a path aligned to the construction direction and any point must be the shortest path along the surface. Given an arbitrary shape, if it can be flattened or unfolded in area-preserving manner with cuts that are orthogonal to the knitting direction, it can be constructed with machine-knitting (without explicit gluing).[4]

[4] Importantly, the flattened surface need not be connected as long as the cuts required to connect the surface are aligned.

Problem Statement 6.2: Transer-Free Patterns. Given a triangulated, disc-like manifold input mesh $\mathcal{M}$ and a knitting direction $\vec{d}$, generate an augmented stitch mesh representation that does not use any faces with transfers.

One approach to come up with such a knitting pattern is to consider a variant of the incremental slicing algorithm presented in section 6.1. Instead of slicing rows, the same idea can be used to slice *columns* on the surface using a geodesic function. Beginning with a chain of edges (*spine*) as a source, a geodesic function can be computed along the entire surface. The *slicing* procedure samples points on a chain at unit distance from the active source. The *linking* procedure links these points to form orthogonal *rows*. the *trimming* procedure removes the processed regions. To account for curvature, short-rows naturally appear since the length of the columns can differ and loops are connected 1-1 yarn-wise. Depending on the initial conditions closed contours may appear and need to be cut to interpret the curve as a column of knit loops.

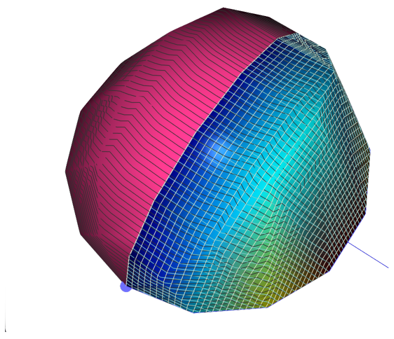


Figure 6.15: Slicing along columns, provides an incremental way to generate patterns entirely comprised of short-rows. The black edges show the discretized row edges and the light edges show the orthogonal gradient direction.

As the stitch size used to generate columns and rows decreases, in the limiting case, each point on the surface is linked (along rows) following the shortest path on the surface to the *spine*. Short-rows begin exactly at the set of points that have more than one unique shortest path to the spine. The set of points that have more than one shortest path to a source form the *cut-locus* or *ridge-tree* with respect to the source. These structures are very well-studied in the field of computational geometry especially for unfolding convex polyhedra (Demaine and O'Rourke [2005]).

Unfolding Polyhedra The star unfolding cuts the polyhedron along geodesics from a source point to lay it out in a star shape around the source point (Agarwal et al. [1997]). In contrast, the source unfolding also uses a source point but cuts the polyhedron along its ridge tree – the locus of points that have more than one unique shortest paths to the source point as shown by Kiazyk and Lubiw [2016]. For convex shapes, it is known that both source and star unfolding generate an overlap free unfolding (Demaine and O'Rourke [2007]). The sun unfolding is a generalization of the source unfolding to geodesic curves as source points (Demaine and Lubiw [2011]).

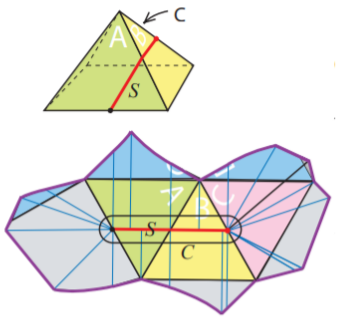


Figure 6.16: A sun unfolding of a convex shape reproduced from Demaine and Lubiw [2011] Figure 2

The key idea that sun unfolding borrows from the source unfolding is the property that the *shortest paths are radially monotone* around the source (for convex shapes and geodesic curves) – and therefore can be laid out in a non-intersecting star-shaped fashion. This property is no longer true for non-convex shapes (with *saddle* vertices that have a total angle greater than 2π). However, it continues to be true that treating the saddle vertex as a (psuedo) source point – all shortest paths that reach it emanate in radially monotone manner or contain another pseudo-vertex strictly further away from the spine source. These paths can no longer be laid out without intersection but they can be ordered.

This sun-unfolding satisfies what is needed to view a flattening as a knitting pattern where the knitting direction is specified by the source path (spine) : points lie along the shortest path to the spine and all shortest paths can be ordered along the spine.

So the task at hand is to compute the sun-unfolding with respect to a spine on the triangulated input mesh and flatten the mesh. To deal with shortest paths and distances on the mesh, we need to compute geodesic distances with respect to the spine. A popular approach to do this for triangle meshes, is by a continuous Djikstra-style algorithm were "visibility windows" are tracked along the edges and propagated forward (Bommes and Kobbelt [2007], Surazhsky et al. [2005]). The windows make it efficient to track all paths that pass the same (portion) of an edge towards the source point. This partioning of the mesh into windows computes the ridge-tree with respect to the input source.

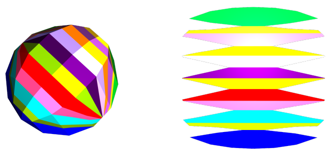


Figure 6.17: Flattening a mesh by cutting along its ridge tree computed from windows tracked for exact geodesic distance computation.

Since any vertex that is not already flat must appear on the ridge tree, for a mesh with many vertices, computing such an explicit ridge-tree can be numerically unstable with many tiny slivers (see figure 6.18 that shows an increasing number of cuts as the mesh approximation of a hemisphere is improved). Further, resolving window propagation when the sources include edges and vertices of the mesh require resolving windows that are circular (from source points) and linear(from source segments) as shown by Bommes and Kobbelt [2007]. To avoid numerical instabilities, we implement a sampling based approach to compute the flattened pattern. We estimate the unfolding as a density map by laying out paths instead of extracting windows or cutting along the ridge tree. This density map needs to be computed only once. It can then be discretized based on the fabrication setup and reinterpreted as knit loops.

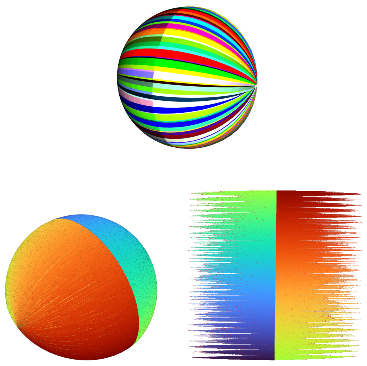


Figure 6.18: (Top) Explicit cutting can include many tiny slivers as the number of (non-flat) vertices in the mesh increases. (Bottom) Paths traced from points sampled on the surface and laid out in monotonic order.

Sampling-based approach

I now describe our approach to computing a sun unfolding for an input surface (figure 6.19).

Input This pipeline takes as input a 3D triangulated manifold mesh and an open, simple, connected path given as a sequence of edges – the spine along which the mesh is to be unwrapped. The only constraint on the inputs is that the mesh is simply connected to the spine.

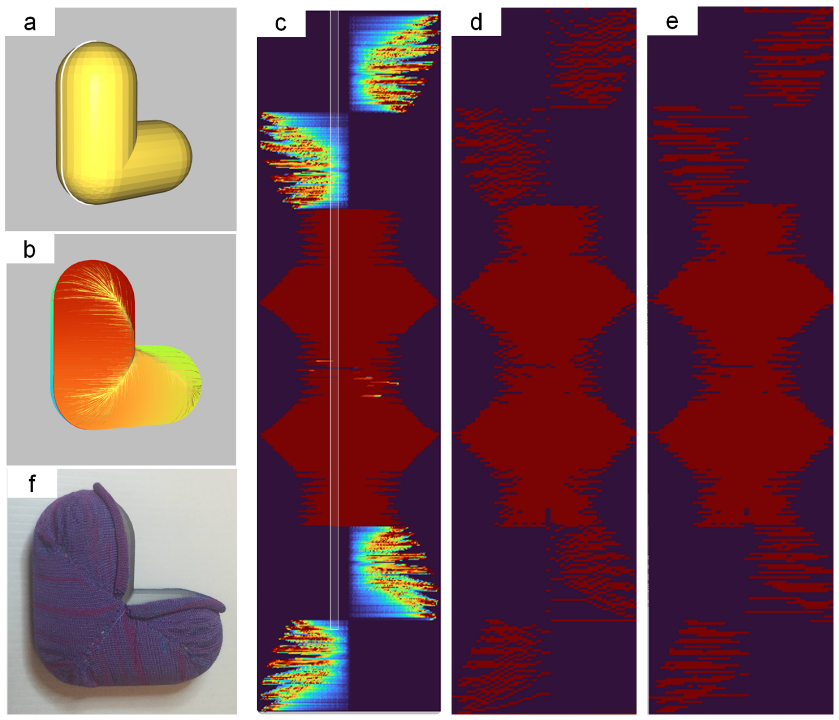


Figure 6.19: a) Given an input triangle mesh and a sequence of edges that define the spine (b) shortest paths are computed to the spine for all sampled points (c) samples are flattened as a density map (d) discretized and (e) optimized with a greedy strategy. (f) The resulting pattern can be interpreted as a short-row only knitting pattern and fabricated on a single bed knitting machine.

Distance function to the spine To compute the shortest path to the mesh, we follow the approach of Surazhsky et al. [2005] to find geodesic distance fields on triangle meshes. Any accurate geodesic distance computation algorithm can be used in this system, as long as computing the shortest paths from points to the spine is not expensive. Each path p is recorded as as a list of list of (psuedo) source vertices that it passes through and distances between them:

$$p = ((v_0, \theta_0, 0), (v_1, \theta_1, l_1), ...)$$

where v_0 is a point on the spine, θ_0 is the angle made with the spine, v_i is a saddle vertex, l_i, is the distance to the previous vertex and θ_i is the angle made with respect to the previous segment.

Mesh triangles are sampled uniformly using barycentric co-ordinates following Osada et al. [2002]. For uniform random variables r_1 and r_2 between 0 and 1, a point is generated with barycentric wieghts $(1-\sqrt{r_1}, \sqrt{r_1}(1-r_2), \sqrt{r_1}r_2)$. The number of samples generated for each triangle is proportional to its area.

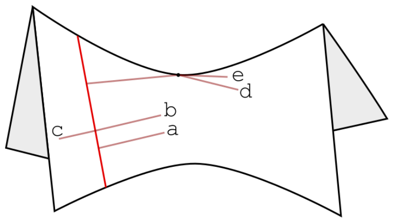


Figure 6.20: Any two paths can be ordered based on their distance along the spine from its boundary point ($a < b$). If the two paths share a common saddle, they are ordered by their angle at the saddle vertex ($b < c$, $d < e$).

Ordering of paths The spine is an open simple edge-sequence which can be laid out monotonically by laying out the edges one after the other along an axis (say the y-axis).

For paths p_i and p_j

$$p_i < p_j \implies d(p_i(v_0), a) < d(p_j(v_0), a)$$

where a is an end point of the spine, $d(.,.)$is distance on the surface

or

$$p_i(v_0) = p_j(v_0) \wedge p_i(v_k, \theta_i) < p_j(v_k, \theta_i) \text{ where } v_k \text{ is the last common saddle} \quad (6.1)$$

Notice that paths from any two points on the mesh can be sorted with respect to the spine. Two paths are either non-intersecting and meet at two distinct points on the spine – in which case they are ordered by their distance along the spine or they meet at some saddle vertex. For two intersecting paths, they are ordered based on the angle made at the common saddle vertex farthest from the spine with respect to the path towards the spine.[5] Figure 6.20 illustrates how paths are ordered in the presence of saddle vertices. Equation 6.1 describes the comparison function used to order paths.

[5] Two distinct paths may reach the spine at the same point (from opposite sides) – in which case the tie can again be broken by the radial angle.

Path unfolding Now that we have an ordering on the paths, layout proceeds in a straightforward fashion:

For each sampled point $s(t_i, b_j, p)$ from triangle t_i at barycentric co-ordinates b_j and associated path p, the layout function maps it to the location (x, y) with a weight α where $x = sgn \cdot \sum_i p(l_i)$. The sign of x is decided based on the orientation of the path with respect

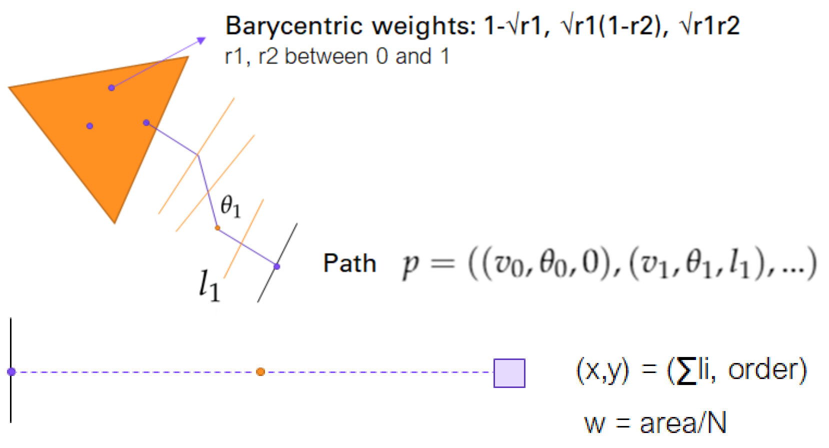

Barycentric weights: 1-√r1, √r1(1-r2), √r1r2
r1, r2 between 0 and 1
θ_1
l_1
Path $p = ((v_0, \theta_0, 0), (v_1, \theta_1, l_1), \ldots)$
(x,y) = (∑li, order)
w = area/N

Figure 6.21: Samples can be ordered and laid out in 2D based on distance and order.

to the spine, clockwise paths are assigned $sgn = -1$, and 1 otherwise. If s is on the boundary, $\alpha = \emptyset$ to mark it specially otherwise $\alpha = area(t)/|samples(t)|$ indicating presence of material weighted by the triangle's area. Although setting $y = r$ rank in order provides a layout, to avoid a very long layout, paths are grouped by their last common saddle vertices into regions, monotonic order within regions (r_g) to order paths as $y = d(p(v_0), a) + r_g + o$ where a is the end point of the spine and o is an offset to avoid overlap with lower regions.

Discretization Given a 2D density map and the stitch size (l_r, l_c), the discretization process rasterizes the density map. First the density map is downsampled to $\frac{W}{2l_r}$ and $\frac{H}{2l_c}$ where W and H are the width and height of the map respectively. In the downsampled density map, a pixel may have full (one) density (indicating presence of material), zero density (indicating a cut region around a convex vertex in the flattening), a value between zero and one(indicating fractional amount of material), or a boundary region[6].

The rasterization strategy is to accumulate density over each column beginning from zero. Once the density accumulated reaches 1, it assigns a pixel to the location, resetting the accumulator. The accumulator is also reset when the pixel location has zero density (indicating a cut region) or a boundary marker (indicating a boundary region).

This descritization can differ from the resampled input density by at most 1 stitch per column, and move 1 stitch per cut to an adjacent region.

[6] To avoid gluing boundary points during fabrication, a secondary yarn can be used as "support material" or the loops can be dropped carefully and new loops be cast on.

```
procedure DISCRETIZE(density, rows, columns, H, W)
    $density \leftarrow resample(density, \frac{rows}{2H}, \frac{columns}{2W})$
    $rows \leftarrow \frac{rows}{2H}$
    $columns \leftarrow \frac{columns}{2W}$
    $fill = threshold(density > 0)$    ▷ For tracking locations where stitches can exist during optimization
    for $c$ in $[0 : columns - 1]$
        $accum \leftarrow 0$
        for $r$ in $[0 : rows - 1]$
            if $density(r, c)$    ▷ not boundary or cut
                $accum \leftarrow accum + density(r, c)$
                if $accum > 1$
                    $density(r, c) \leftarrow 1$
                    $accum \leftarrow accum - 1$
                else
                    $density(r, c) \leftarrow 0$
```

Algorithm 1: Psuedocode for discretization. Given a density map of size $rows \times columns$ and stitch size as $W \times H$ generate a binary stitch pattern.

The discretization procedure is described in pseudo-code listing 1. However the output may include very short fragments along the rows since the accumulation process proceeds over the columns. To remedy this, our system greedily optimizes the average length of the fragments (contiguous pixels along the rows). Pixels are re-assigned only within regions of non-zero density to avoid moving pixels across a cut region. In each iteration, the sub-fragment that best improves the objective function (average row length) is selected and updated. The width W of the pattern is fixed, therefore the maximum average length is bounded. The optimization stops when no further updates can be made. Pseudo-code 2 describes this algorithm. Figure 6.22 illustrates the optimization procedure for a simple pattern.

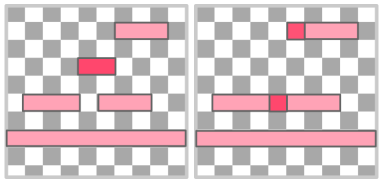


Figure 6.22: A greedy optimization is used to improve the discretization so that the average length of the rows is improved. The dark row on the left is moved to improve the overall edge length in two iterations in this illustration.

This discretization approach can be further improved by considering both the rows and columns while accumulating densities using a seam-carving like approach. We upsample along columns by doubling the columns (avoiding failure-prone single stitch situations). This discretized graph can be viewed as *knit-graph* over pixel nodes, row-wise and yarn-wise connections based on adjacency. The downsampling along rows can be undone by the tracing procedure described in chapter 6, since each row is visited exactly twice and creates a single continuous path along the nodes(stitches) and faces can be assigned to construct the dual augmented stitch mesh.

The incremental remeshing based strategy and the sampling based unfolding based strategy are fundamentally similar – differing in when and how discretization is performed. However, the connection to un-

Algorithm 2: Psuedocode for greedy optimization. The greedy optimization procedure considers all sub-intervals that can be moved upward or downward to strictly improve the average row length. In each iteration, the best improvement is greedily picked and the process is repeated until no more improvements can be made.

```
1:  **procedure** GreedyOptimizeRowLength(density, fill, rows, columns)
2:      $intervals \leftarrow computeRowIntervals(density)$
3:      $average \leftarrow avgLen(intervals)$
4:      **while** true
5:          $move \leftarrow \emptyset$, $best \leftarrow average$
6:          **for** $r$ in $[0 : rows - 1]$
7:              **for** $[i, j]$ in $intervals[r]$
8:                  **for** $r'$ in$[r + 1 : rows - 1]$                ▷ Move up
9:                      **if** $density[r', i : j] = 0 \wedge fill[r', i : j] > 0$
10:                         $l \leftarrow avgLen(intervals \setminus [r, i : j] \cup [r', i : j])$
11:                         **if** $l > best$
12:                             $best \leftarrow l$
13:                             $move \leftarrow [[r, i : j], [r', i : j]]$
14:                     **else** break
15:                 **for** $r'$ in$[0 : r - 1]$     ▷ Move down     ▷ Repeat 10-15
16:         **if** $average = best$
17:             **return** $density$
18:         $average \leftarrow best$
19:         $do(move)$
20:         $intervals \leftarrow computeRowIntervals(density)$
```

folding techniques and fabrication techniques with constrained ways to glue cuts can lead to interesting patterning ideas. For example, the orthogonal constraints of the *weft* and *warp* yarns in weaving introduces the same constraints as the short-row-only knitting problem. A flattened pattern can be reinterpreted as a *weaving* pattern by discretizing with the appropriate unit size. Crochet being a hand-crafting technique has far fewer constraints than machine knitting or weaving. However, the flattened results can be interpreted as a subset of simple crochet patterns. Figure 6.23 shows a simple box pattern visualized with knitting, weaving and even crochet. Apart from textiles, even paper-craft and cut-and-sew patterns might benefit from exploring constrained gluing directions as a way to automate the fabrication process with a simpler setup. Fabricated results using the remeshing

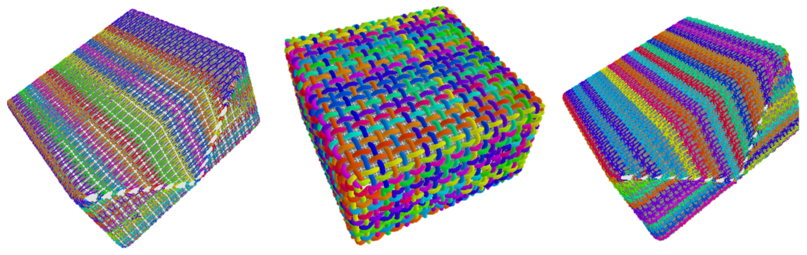


Figure 6.23: A transfer-free augmented stitch mesh generated using the sampling approach interpreted as knitting, plain weaving and crochet.

techniques proposed in this chapter are shown in chapter 8.

7
Turning Knitting Patterns into Machine Code

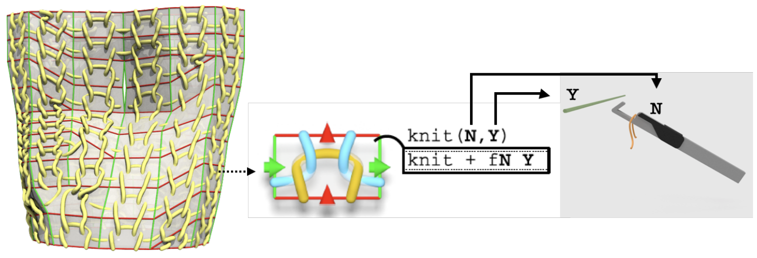


Figure 7.1: Scheduling assigns bed-needle allocations for each stitch in an augmented stitch mesh in order to execute its face program.

The augmented stitch mesh created from scratch or from a 3D model represents a machine knitting pattern including the knitout code fragments that describe how each face must be constructed locally. However, at this instance, this representation is *unscheduled* – information on *where* these faces must be constructed on the machine and how loops must be moved to these locations is unknown. This separation was useful in supporting stitch mesh creation and editing operations in a general 3D space instead of the machine space. However, to finally execute the pattern on the machine, *needle*, *time* and *layer* information for each stitch face needs to be assigned and passed on to the face programs. Edge connections indicate where a produced resource must be moved for consumption. However, the mesh does not directly include instructions required to move resources around. Any necessary loop transfer, yarn miss and machine rack operations need to be interleaved in the instruction stream to ensure stitches are moved to the correct location when edge resource assignments differ. In this chapter, I present a user-in-the-loop scheduling system for general augmented stitch meshes and an automatic scheduling system for manifold augmented stitch meshes that uses two layers.

A complete construction plan for a knit object on a two-bed knitting machine that encodes all its resource constraints can be represented by an *embedded instruction graph*. It consists of :

$\mathcal{P} \equiv \{p_1, ...\}$	Positions(Bed Needle locations)
$\mathcal{Y} \equiv \{\emptyset, y_1, ...\}$	Yarns (Carriers)
$\mathcal{L} \equiv \{\emptyset, l_1, ...\}$	Loops
$\mathcal{N} \equiv \{n_1(t, op, in := \{(y_i, p_a), .., (l_j, p_b), ...\}, out := \{(y_i, p'_a), ..., (l_j, p'_b)\}), ...\}$	Instruction Nodes
$\mathcal{R} \equiv \{(n_i, n_j), ...\}$	Directed resource edges
$\mathcal{S} \equiv \{(l_i, l_j, s_{ij}), ...\}$	Slack edges

Instruction operation: An instruction node $n(in = \{(resource, position)\}, out = \{(resource, position)\}) \in \mathcal{N}$ consumes zero or more loop and yarn *resources* at specified *positions* in *in* and produces them at their output positions in *out* at time t. Positions $(\emptyset, .) \in in$ must be empty before the instruction operates and positions $(\emptyset, .) \in out$ are empty after the instruction executes.

Resource Alignment: $(n_i, n_j) \in \mathcal{R}, (r, p_i) \in n_i^{in}, (r, p_j) \in n_j^{out}, r \neq \emptyset \implies p_i = p_j$ and instruction n_i happens before instruction n_j i.e., $n_i^t < n_j^t$.[1]

[1] n_i^t is used to reference property time t of node index i.

Slack maintenance: For $(l_a, l_b, s_{ab}) \in \mathcal{S}$ and $n_i, n_j \in \mathcal{N}$ such that $(l_a, p_a) \in n_i^{in/out}$ and $(l_b, p_b) \in n_j^{in/out}$, $|p_a - p_b| \leq s_{ab}$.

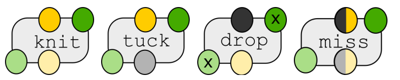
This graph relates to information from the augmented stitch mesh as follows:

1. All loops $l \in \mathcal{L}$ are produced by face program instructions.
2. Each face program instruction has an associated instruction node $n \in \mathcal{N}$.
3. Each connection between faces is associated with zero or more instruction nodes to perform the connection by transfers.
4. The slack attribute for any pair of loops generated within a face program is *set* equal to the maximum distance between its yarn-wise connected positions in that face.
5. The slack attribute for yarn-wise connected loops between different faces is *set* equal a uniform constant for the entire object i.e., the embedded graph constructs the augmented stitch mesh object with uniform density (unless specified otherwise by the face program).
6. Any total order of the embedded graph i.e., an ordered list of instruction nodes $n_1, n_2, ...$, such that $n_i^t \leq n_{i+1}^t$ is topologically equivalent to the augmented stitch mesh.

Visualizing the embedded instruction graph The embedded visual graph provides a time-line view of the construction process. We visualize each instruction node as block that spans needle locations present in the node. Needles are identified by the x-coordinate and node time by the y-coordinate of the position. Incoming loops are identified as 🟡 and yarns as 🟢, outgoing loops and yarns are identified by 🟡 and 🟢 respectively. Empty incoming locations for an instruction are visualized as ⚪ and similarly locations left empty after executing the instruction are shown as ⚫. An empty circle is used to indicate that the operation does not affect a particular location – the last resource placed in that location continues to remain in that location.

This representation can be viewed as a general and more fine-grained version of the needle-bed scheduling system proposed in McCann et al. [2016] and a augmented version of the *loop-view* in SDS-one systems with an explicit view of resource constraints.

An instruction node that creates a loop by knits[2] or tucks[3] is illustrated below – explicitly highlighting the dual between a knit stitch face (with a single instruction) and the graph.



[2] For example, $n_1 = \{(t, op = \texttt{knit}, in = ((l_1, f1), (y_1, f1_l)), out = ((l_2, f1), (y_1, f1_r))\}$

[3] $n_2 = \{(t, op = \texttt{tuck}, in = ((l_1, f1), (y_1, f1_l)), out = ((l_1, f1), (l_2, f1), (y_1, f1_r))\}$ or $n_3 = \{(t, op = \texttt{tuck}, in = ((y_1, f1_l)), out = ((l_1, f1), (y_1, f1_r))\}$

The resource locations might hold stacked loops that can be explicitly recorded. For instance, a tuck instruction that stacks a loop onto a needle location requires the location as an input and output resource whereas a tuck on an empty location explicitly requires the resource to be empty. Loops cannot be separated from a needle location – so this information can be dropped and viewed as one or more stacked loops as shown on the right (figure 7.2).

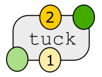
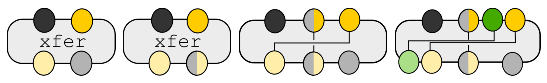
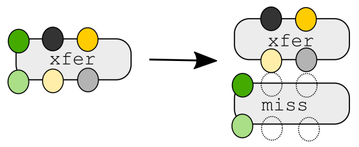


Figure 7.2: Loops are placed at resource locations, which means stacked loops appear at the same location and can be annotated numerically.

Transfer instructions that move loops around at some *rack* value encompass *all* the loops and yarns that exist within the racking span of the source and target locations to visualize the local layering of the loops – this can be viewed as a *braid* recorded between the source and target resources.



When transferring loop that is attached to the yarn carrier, the yarn location also changes. For clarity, such operations can be split into a transfer operations strictly over loop resources and a miss operation over a yarn resource as shown below:



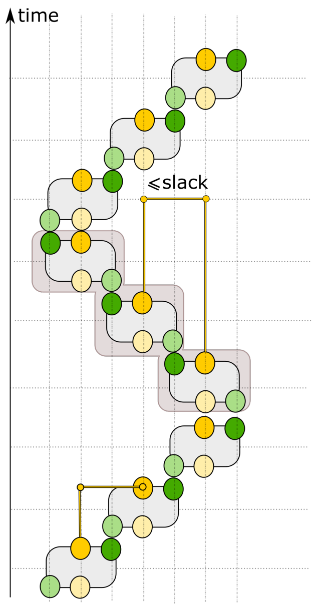


Figure 7.3: A simple knit sheet laid out as an instruction graph.

These node instructions can be placed in a 2D layout according to their *time* (n^t) and resource location values along a timeline. Resource connections are explicitly seen by alignment along the timeline. Resource edges that are not strictly vertical in the visualization highlight an alignment failure. A set of blocks that do not overlap in time, can be aggregated into a *pass* visually shown by the shaded set of blocks in figure 7.3.

A sequence of transfer instructions can be aggregated into a single *move* operation for abstraction. Such a move operation encodes within it the braid word produced by the transfers over the loops and yarn operations. The internal temporary loop locations used by the transfer operations are also marked as required empty locations.

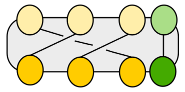


Figure 7.4: A move operation that encompasses some sequence of transfer operations encoding a braid of its loops.

These move operations can be viewed as an instance of a planning problem, they can be replaced by a different sequence of transfers as long as the braid word produced is identical.

Constructing the embedded graph Given an augmented smobj that has a monotonic order consistent with a co-ordinate axis, the two orthogonal axes are treated as a layer axis and a needle axis. Each face is assigned a single layer value and is assigned a needle location for all its boundary resources. The smobj is uniformly scaled up such that no two faces occupy the same needle location.

A connection between two resource locations that do not occur at the same location can be viewed then as *moving* the resource at the source location of the connection to the target location. Because such movements have to be introduced by *transfer* and *rack* instructions, this edge is rasterized and is reinterpreted as a sequence of move operations between layers at the same location or between needles in the same layer. Each move operation can be interpreted as one or more transfer operations. The local layering information of the faces can be used to ensure that loops from different layers are correctly separated (described in section 7.1.3).

These instruction nodes serve the purpose of recording the topology of the input surface.

The resource-planning problem The scaling operation employed to space stitch faces implies that the distance between resources may not match the distance implied by the input mesh. Second, the number of layers may be too high to effectively produce on any reasonable machine. However, the *intended* metric value can be recorded for each connection by assuming any two loops connected by a yarn-wise edge between stitch faces have a unit slack value[4]. This embedded graph provides a schedule from which the correct topological object can be constructed with (potentially) high geometric distortion and a sub-optimal number of layers. For a schedule that matches the geometric intent of the smobj, the slack edges between resources and their location need to maintain the recommended slack value.

[4] The number of layers is factored into the unit value to account for the interleaving placement of layers on the physical bed

The overall scheduling problem can therefore be expressed in two parts:

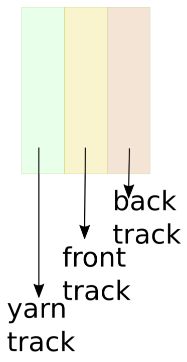
Problem Statement 7.1: Scheduling.

Embedding Given an augmented stitch mesh pattern, generate an embedding – an assignment of needle, layer and time values for each face instruction that describes *where* a loop must be produced – in the form of an embedded graph that is *topologically* equivalent to the stitch mesh.

Planning Given an embedded instruction graph, rewrite the graph (by inserting transfer and miss operations) such that all slack edges maintain the correct slack value while preserving the topology of the input graph.

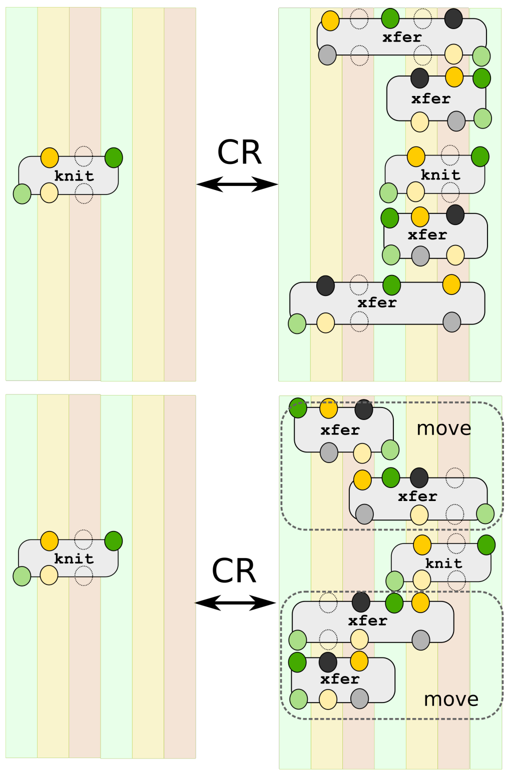
7.1 *User-in-the-loop embedding for general structures*

Given an embedded graph extracted from an stitch mesh, we will first focus on the problem of eliminating excess slack. We do this by editing the embedded instruction graph with three simple rules. The graph is laid out with a background scheme shown on the right to represent the interleaved front bed, back bed and yarn track.



1. Conjugate with transfers. (CR, CL, CO) Any instruction can be conjugated with paired [5] transfer operations to move the instruction the left(CL) or the right(CR) or across(CO) locally. We only consider transfers at 0,1 and -1 racking for minimizing the number of primitive operations. This can be generalized based on the racking value of the machine. The paired transfers can have two forms based on when racking occurs:

[5] When conjugating a loop making instruction node such as `knit`, a transfer to the opposite bed before construction would disrupt the loop-to-loop connection (and more subtly would change the topology of the yarn), to avoid this transfers are always paired when conjugating i.e they introduce a pure translation. A group of transfer operations can be abstracted as a single 'move' operation to reduce the number of operations viewed.



Conjugation can also be executed across by moving the loop to the opposite bed track, in this case, yarn positions are reversed to ensure that the correct loop topology is produced.

Applying CR on instruction node n_a of an embedded graph $G(\mathcal{N})$ produces $G'(\mathcal{N}')$ where[6]:
$\mathcal{N}' \leftarrow \mathcal{N} \cup \{n'_{a1-}, n'_{a2-}, n'_{a1+}, n'_{a2+}\}$
$n'_{a1-} \leftarrow \{t := n^t_a - 2, op := \texttt{xfer}, in := n^{in}_a, out := opp(n^{in}_a) + 1\}$
$n'_{a2+} \leftarrow \{t := n^t_a - 1, op := \texttt{xfer}, in := opp(n^{in}_a) + 1, out := n^{in}_a + 1\}$
$n'_{a1-} \leftarrow \{t := n^t_a + 1, op := \texttt{xfer}, in := n^{in}_a + 1, out := opp(n^{in}_a) + 1\}$
$n'_{a2+} \leftarrow \{t := n^t_a + 2, op := \texttt{xfer}, in := opp(n^{in}_a) + 1, out := n^{out}_a\}$
${n'_a}^{in} := n'_{a-}out$, $n'_a out := {n'_{a+}}^{in}$
and,

$\forall n' \in \mathcal{N}'$ s.t $n'^t < n^t_a, n'^t := n^t - 2$
$\forall n' \in \mathcal{N}'$ s.t $n'^t > n^t_a, n'^t := n^t + 2$

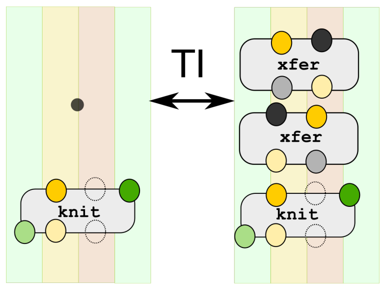
The pre-condition on $G(\mathcal{N})$ needed to apply CR can be expressed as: $\nexists l$ s.t $(l, p) \in n^{out}_1$, $n^t_1 < n^t_a, (l, p) \in n^{in}_2$, $n^t_2 > n^t_a$ and $(x \neq l, p) \in opp(n^{in}_a) + 1$ for $n_1, n_2 \in \mathcal{N}$.

[6] $n^{in} + 1$ is used as a short-hand to increment position values of all resources by 1. Similarly $opp(n^{in})$ refers to the opposite bed for all resources in n^{in}

The operations introduced does not place a resource in a location which overlaps a different resource.

2. Insert or Suppress inverse transfers (TI).

A pair of inverse transfer operations can be introduced at any time t. Inversely, a sequence of transfer operations that do not change the location of the loops and yarns can be eliminated.



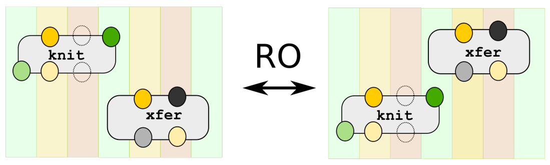
Applying $TI(r)$ at rack r, after node n_a^t to an embedded graph $G(\mathcal{N})$ produces $G'(\mathcal{N}')$ where:
$\mathcal{N}' \leftarrow \mathcal{N} \cup \{n'_{a1+}, n'_{a2+}\}$
$\forall n' \in \mathcal{N}'$ s.t $n'^t > t, n'^t := n^t + 2$ and
$n'_{a1+} \leftarrow \{t := t+1, op := \texttt{xfer}, in := n_a^{out}, out := opp(n_a^{out}) + r\}$
$n'_{a2+} \leftarrow \{t := t+2, op := \texttt{xfer}, in := opp(n_a^{out}) + r, out = n_a(out)\}$

A similar pre-condition for $TI(r)$, $\nexists l$ s.t $(l,p) \in n_1^{out}$, $n_1^t < n_a^t$, $(l,p) \in n_2^{in}$, $n_2^t > n_a^t$ and $(x \neq l, p) \in opp(n_a^{in}) + r$ for $n_1, n_2 \in \mathcal{N}$.

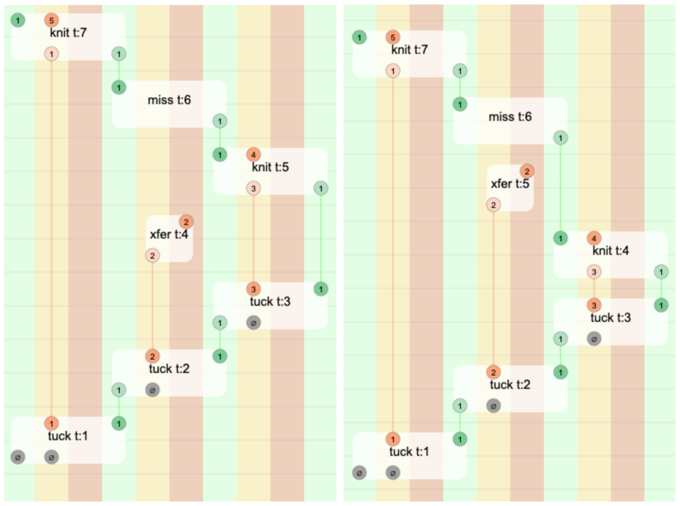
Inversely, given a sequence of instruction nodes $n_a, ...n_{a+k}$ such that $n^{op} = \texttt{xfer}$, $n_i^{out} = n_{i+1}^{in}$, and $n_a^{in} = n_{a+k}^{out}$, the nodes can be deleted to produce a new graph $G'(\mathcal{N}') = \mathcal{N} \setminus \{n'_a, ..., n'_{a+k}\}$ where:
$\forall n' \in \mathcal{N}'$ s.t $n'^t > n_{a+k}^t, n^t := n^t - k$

3. Reorder independent nodes $RO(t)$.

Two time-adjacent instruction nodes that do not have overlapping resource spans, can be reordered.



Applying $RO(t)$ on $G(\mathcal{N})$ with independent instruction nodes n_a and n_b such that $n_a^t = t$ and $n_b^t = n_a^t + 1$ produces $G'(\mathcal{N})$ where $n_a'^{\,t} := n_b^t$ and $n_b'^{\,t} := n_a^t$



Independent xfer and knit nodes at time 4 and 5 can be reordered as shown. Nodes (miss and xfer) at time 5 and 6 cannot be reordered since their spans overlap. The graph on the right shows the result after applying $RO(4)$ to the graph on the left.

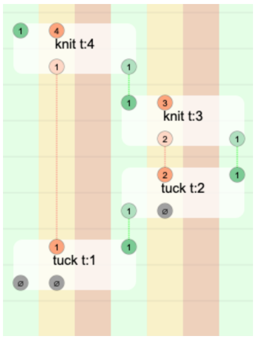
Each of these rules can be applied as long as the resource constraints are maintained (i.e., a transfer operation introduced by the conjugation can only be applied if the new locations were empty or already shared the same resource. By definition, the rules not change the topology of the underlying object, any loop movement is immediately followed by undoing the operation. Therefore the time at which a slack change occurs (increase or decrease) can be shifted. By combining conjugation, paired transfers and inverse deletions, effectively slack can be reduced. By repeatedly using these rules, we iteratively eliminate slack. This process can be manipulated by an end user interactively by directly applying these rules to improve the search process.

Composite operations for shifting A sheet made with two rows of two loops each can be shifted by one to the right performing the following operations (as shown in figure 7.1): Apply CR to each loop on the right most column in time order. Apply TI to eliminate intermediate transfers. Apply CR to each loop on the left most column in time order. Apply RO to bring transfers together and TI to eliminate intermediate transfers. Note that slack is violated at the end of step 2, however this violation can be tracked and is fixed by the end of step 4.

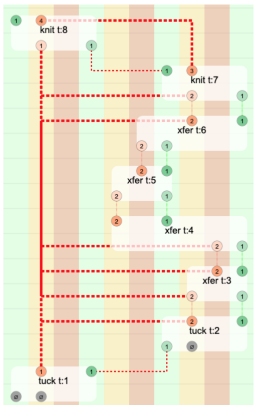
Composite operations for rotation Similarly, to rotate the sheet instead of moving loops to the right it is moved across by applying CO, in time order and using TI to eliminate redundant transfers as illustrated in figure 7.2. Notice that the yarn direction is correctly modified to maintain the correct orientation. The left most column can be shifted to maintain the yarn slack.

Grouping nodes As a convenience, any sequence of nodes can be grouped and viewed abstractly as a single instruction node that produces all the

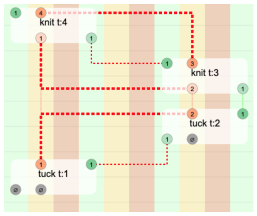
Table 7.1: Translating the graph to the right by applying a sequence of graph transformation rules. Slack edge violations are visualized by thick red dashed lines



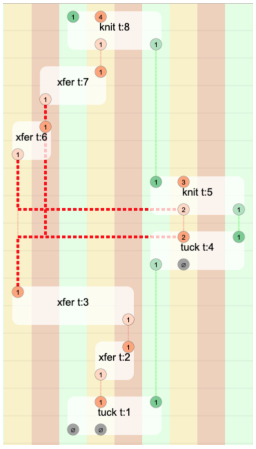
(a. Initial)



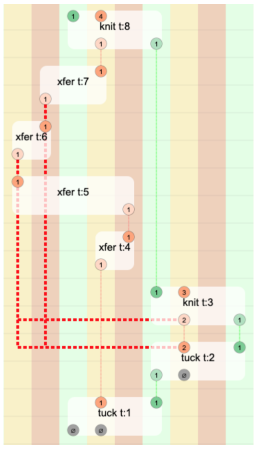
(b. Apply CR)



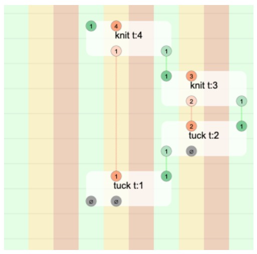
(c. Apply TI)



(d. Apply CR)

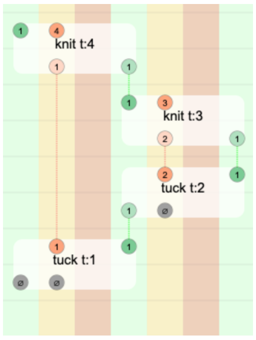


(e. Apply RO)

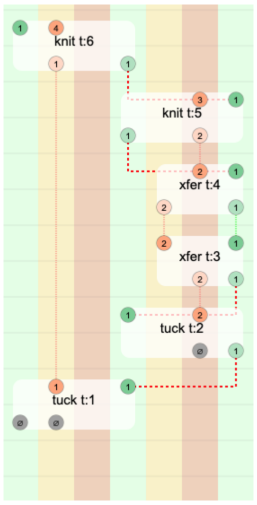


(f. Apply TI)

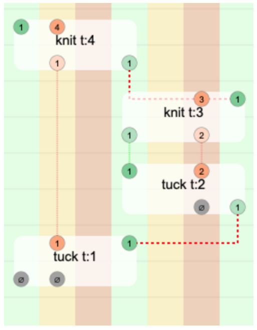
Table 7.2: A sheet is rotated by applying a sequence of transformation rules.



(a.)



(b. Apply CO)



(c. Apply TI)

loops produced by the sequence and consumes all the loops consumed by the operations. Rules can be applied to a node group just as they are applied to single node. To check the condition on tracks being empty (or have the appropriate resource), locations within the group need to be tested. Because the group of nodes is moved together, no slack changes are introduced between resources in the group within that time when a rule is applied.

7.1.1 Slack handling

The embedded graph includes slack edges that annotate the *desired* slack between two loops factoring in the number of layers. Construction slack (slack between a resource and any other resource at a time when it first appears) must be exactly met whereas slack between loops after their construction must not exceed the associated limit. While modifying the embedded graph into one with slack constraints respected, our system must ensure that no topological changes are introduced. The rules described above when applicable, do not introduce a topological conflict (although they can introduce more slack conflict).

User-in-the-loop Search Procedure The adhoc search procedure shown for composite rotation and shifting illustrates that more slack violations can be introduced while attempting to fix slack elsewhere. To simplify the number of operations needed to be searched, the user can group instruction nodes temporarily and apply a single rule to the entire set. To be able to systematically search the space, rules need to be applied in an order such that additional violations can be contained. Coming up with an automatic system to update the graph is an important future work that would be needed to schedule large objects.

There are a few situations where the rewriting the graph for slack satisfaction can not resolve all slack issues. We have already encountered this situation when *unbalanced* rows appear in the knit graph and stitch mesh (Property 6.6). In situations where the augmented stitch mesh layout is unbalanced, the user can chose to manually resolve the situation by a)relaxing the slack constraint by increasing the number of layers b) by changing the face program to add more slack to the loops or c) edit the augmented stitch mesh to generate a balanced structure.

7.1.2 Embedding the augmented stitch mesh

I now describe our setup to convert an input augmented smobj into an embedded form. Embedding a general augmented stitch mesh requires maintaining the topology of the shape being described, the alignment of the loop and yarn edges as well as the layer depths. A single face program describes a way to construct a small sheet like portion of the surface. Layers effectively segment the surface into sheet-like patches. Each face is assigned to one layer in our augmented stitch mesh definition. [7]

[7] To extend this framework for multi-layer faces, each face may be assigned to one or more *consecutive* layers using an offset value.

The edges of each face can be associated with needle *resource* locations on the machine bed. Within the layer, the resources that these face edges would occupy should agree with the template face program associated with the face. The embedding must also maintain the *order* implied by the edge directions on the stich mesh – yarn-wise and loop-wise previous loops must be constructed before the loops that consume them.

Instead of performing this in a fully automated way, here we allow the user to annotate the augmented stitch mesh with hints that help generate the embedding.

The scheduling system allows the user to annotate the mesh in the following ways:

Layer annotation associates a layer number with a face index in the augmented stitch mesh. All instructions in the face program are interpreted using this index as the front-most layer.

Resource annotation associates a needle number with a face and edge index in the augmented stitch mesh. The face program and layer information are combined to establish the physical *bed-needle* location for the resource.

Order annotation associates a happens-before constraint between two face instruction indices. This allows the user to influence the topological order on the DAG when multiple such orders exist.

These annotations are propagated to adjacent faces using edge-to-edge connection using a simple propagation scheme when there is no conflict introduced. In case of a conflict, the user is required to generate annotations to resolve conflicting cases. Given a fully annotated mesh, resource annotations and layer annotations are interpreted as a needle and layer function. A topological order of the underlying DAG is computed by adding constraints from the order annotations, to generate a time function. The rest of the pipeline maintains topological equivalence to this input augmented smobj specified with a monotonic embedding in time or with order annotations to construct the intended DAG.

Given an embedding, edge connections may not agree along yarn edges and loop edges for two reasons: a) The embedding requires loops to shift in order to match the global topology of the shape b) The stitch mesh includes faces that involve transfer operations em Before a face instruction is executed, if the edge connections disagree on locations(resource or layers), transfer instructions need to be executed. These operations need to ensure that *slack* – the distance between yarn-wise adjacent loops – is maintained within the prescribed slack limits. The edges are viewed as a linear connection between faces that are rasterized when constructing the embedded graph. This rasterization procedure may introduce more layers, that are eventually reduced during resource planning.

7.1.3 Maintaining layers during construction

A multi-layer setup is realized by *interleaving* the layers physically on the two beds of the machine. The scheduler ensures that this interleaving of layers is maintained throughout the execution of the object.

Face programs associated with the augmented stitch mesh and transfer instructions generated describe the operation in isolation within a layer. When chaining face programs together in such a setup, these layers can get *tangled* destroying the depth-ordered interleaving.

Tangling. Tangles can be introduced for two reasons:

Loop captures yarn Such a tangle can occur when loops on the back are front knit while loops on a front layer are present in a back holding positions or vice-versa.

Yarn captures yarn Such a tangle occurs when a frontwards yarn crosses a backwards yarn during construction or vice-versa.

The scheduling system needs to ensure that these operations can be executed without tangling.

To prevent captures by loops, layer ordering is explicitly maintained before every operation that creates or moves loops. Every face program instruction is re-interpreted in the context of the layer in which it is running and the current state of the interleaved machine beds.

To prevent captures by yarns, yarns are first assigned in a way compatible with layer ordering. Then, yarns are temporarily moved to safe locations before every operation to avoid any potential tangle. These movements can be executed using the `miss` operation, that moves the yarn to a specific needle location as if to construct a loop and parks the yarn with respect to that location: If a back(front) yarn crosses a front(back) yarn in the rightwards(leftwards) direction, the front(back) yarn is moved further rightwards(leftwards) by inserting a `miss` instruction.

The effective interleaved needle position for each loop i is calculated based on the total number of layers L and its layer l_i as $i' = L \cdot i + l_i$.

A face instruction `op D N CS` is effectively re-interpreted as follows:

1. kick yarns Y not in CS using miss D fN' Y if $P(Y) < N'$ and $P(CS) < P(Y)$ for $D = +$ and vice-versa.
2. move loops i, $l_i < l_N$ to the front using xfer bi' fi' if i' is on the back.
3. move loops i, $l_i > l_N$ to the back using xfer fi' bi' if i' is on the front.
4. perform operation `op D` N' `CS`

For the restricted case of a two-layer machine and tube-like surfaces, given the strong constraint on the topology, a full embedded graph can be constructed directly.

7.2 *Explicit embedding of tubular layouts for two-layer machines*

This restricted case of a two-layer machine allows us to use the following constraints to simplify the scheduling problem:

1. Balanced Cycles Any tubular section must be laid out in a balanced

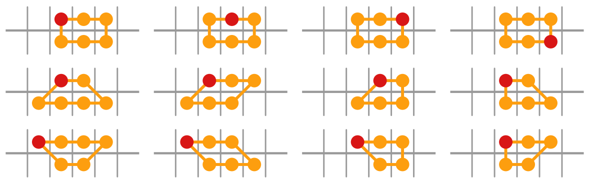


Figure 7.5: Valid balanced layouts for small cycles.

way. Each tubular section must therefore use both the layers – layers correspond to the front and the back of a tube as embedded on the machine. At a splitting or merging event, these layouts must agree.

2. Upward Planarity

For a valid embedding, no two cycles can move past or switch relative ordering with respect to other cycles on the machine. No crossing event (described in Figure 4.1) can occur in the skeleton or graph of this embedding.

For a knitting program to form a desired set of stitches, it must respect the yarn-wise and loop-wise dependencies of those stitches as recorded by the augmented stitch mesh data-structure. We can restate the resource alignment dependency properties in terms of machine execution with:

Property 7.1: Order. All stitches must be constructed in the yarn-wise order specified by the pattern. All loops that a stitch depends upon must be constructed before it and must be available on a needle at the time of constructing the stitch.

Property 7.2: Adjacency. All yarn-wise adjacent stitches must be constructed on adjacent needles. (Aligned needles across layers are also considered adjacent.)

Our scheduler turns each face into one more or *face instructions* – low-level items that, together, produce and consume the same number of loops as a face – and breaks the sequence into logical *passes*. These face instructions serve as a placeholder for the faces, and allow our system to decouple scheduling for storage locations from the operations performed on those locations. The face instructions are maintained as an *instruction graph* with logical directed edges between instruction dependencies. Passes are constructed based on the following conditions:

- All face instructions in a pass have the same direction.
- A pass does not have more than a limited number of "increase" or "decrease" shaping operations that change its width.

Pass *a* is said to depend on a pass *b* if pass *a* uses the loops produced by pass *b* (i.e., it reads from storage locations last written to by pass *b*). Each pass may be dependent on zero or more passes. If a pass depends on exactly one previous pass and exactly one pass depends on it, it is referred to as a *regular* pass and the rest are *critical*. These critical cycles decompose the shape into tubular segments.

Once passes are constructed, the stitches need to be mapped to machine locations. An upward planar embedding is identified by enumerating *all* embeddings of the critical passes. Based on the shape of these critical passes, intermediate passes are filled in by assigning a shape that minimizes loop movement. Finally, needles and layers are assigned to all loops using the computed shapes, and instructions can be generated.

Exhaustive enumeration of critical cycles could become a prohibitive space to explore. The key insight that makes our scheduling algorithm possible is the observation that at connections *between* segments, loops must obey a valid cyclic layout both for the segment that produced them and the segment that will consume them (Fig 7.5). The restriction to tube-like layouts makes the number of valid cycles to enumerate feasible.

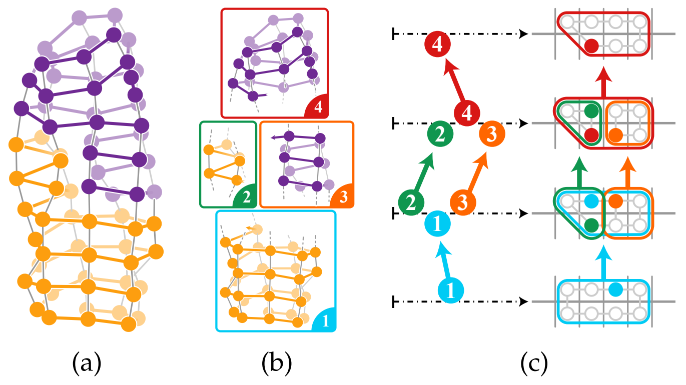
Therefore, the next step of scheduling is for our system to select a layout for each connection between tubes. These connection layouts are chosen to minimize a heuristic cost defined over the segments that tries to reduce transfers, and so that they do not force any segments to cross during knitting (Figure 7.6c). The number of connections between segments is small enough that a greedy enumeration of layouts for these connections terminates quickly – either by exhausting all options or by finding a valid assignment. Once layouts for the connections between segments have been determined, a layout for each step is computed using the same heuristic.

Starting from the instruction graph, our system first identifies sequences of consecutive stitches that take place on a generalized cylinder. This segmentation is accomplished by tracking the connections between loops currently held on the machine bed and performing splits/merges when a loop-wise parent of a stitch does not appear next to the yarn-wise previous stitch in the current cylinder.

At any point during the construction of one of these *segments* of stitches (Figure 7.6b), loops from that segment must be held on the machine bed in one of a small number of layouts (Figure 7.5). Specifically, for a cycle of N loops there are $5N$ (for N even) or $4N$ (for N odd) possible layouts. In general, the winding direction of a cycle cannot be changed through transfers, so our system only considers layouts in one winding direction.. We show layouts using a dotted outline to indicate the needles occupied by the cycle and a dotted circle to indicate the location of a designated loop in the cycle. Conveniently, the transfer planning algorithm of McCann et al. [2016] can move cycles between any two layouts; including adding or removing loops for increases and decreases.

The key insight that makes our scheduling algorithm possible is the observation that at connections *between* segments, loops must obey a valid layout both for the segment that produced them and the segment that will consume them (Property 6.6).

Therefore, the next step of scheduling is for our system to select a layout for each connection between tubes. In effect, our system is determining an upward planar embedding of the Reeb graph of the object, as discussed in chapter 4.



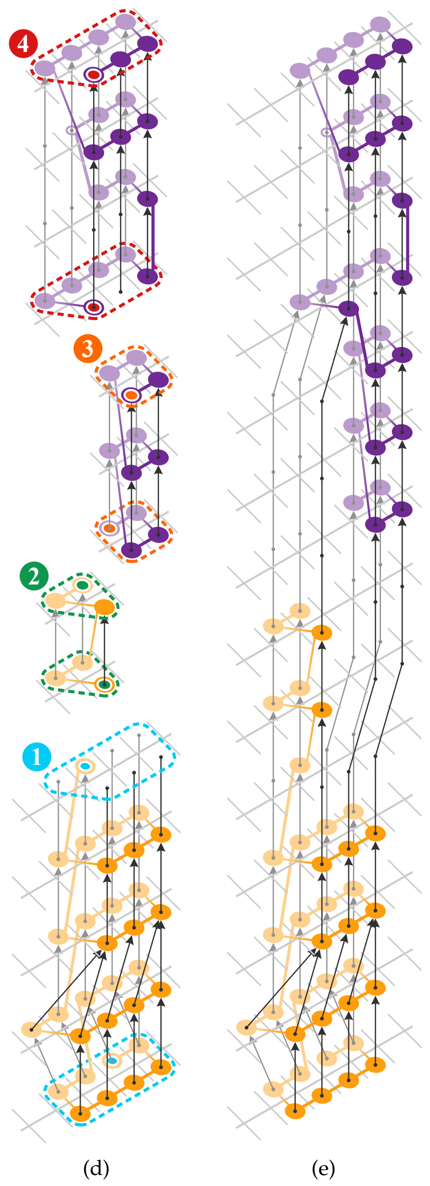


Figure 7.6: Given (a) input instruction graph, our system (b) divides them into tube-like segments, (c) determines layouts for the connections between segments and their left-to-right ordering, (d) determines layouts for each construction step within each segment, and (e) combines the segments into a final knitting program.

Layouts are selected base on, in order of importance: whether the shape has aligned front and back layers (only possible for even cycles), how many loops must change layers between the beginning and end of the segment, and how many loops must shift left or right between the beginning and end of the segment. In the two layer case, the front layer is assumes even needles on the front *bed* and the back layer assumes odd needles on the back *bed* hence layers and beds are used interchangeably. Connection layouts are selected to minimize (lexicographically) the following tuple of per-segment costs:

$$\sum (\mathrm{u}(L_{\mathrm{s}}) + \mathrm{u}(L_{\mathrm{e}}), \mathrm{r}(L_{\mathrm{s}}, L_{\mathrm{e}}), \mathrm{s}(L_{\mathrm{s}}, L_{\mathrm{e}}))$$

Where the terms are defined as follows:

- $\mathrm{u}(L)$ (alignment): 0 if the shape is aligned on the front and back bed (only possible for even cycles) and 1 otherwise
- $r(L_s, L_e)$ (roll): the number of loops that must change beds between the beginning (L_s) and end (L_e) of the segment
- $s(L_s, L_e)$ (shift): the number of loops that must shift left or right between the beginning and end of the segment.

The position of a loop at the beginning of a segment is considered to be the position of its earliest column-wise ancestor in the same segment.

Our system optimizes connection layouts, which imply the starting and ending layouts of *connected* segments, while this cost is defined over the starting and ending layout of each *individual* segment.

Once layouts for the connections between segments have been determined, our system finds an optimal layout for every construction step in each segment, constrained by the already-assigned starting and ending layouts (Figure 7.6d). Here, again, optimal means that layouts should be aligned, loops should not switch beds, and loops should not shift left or right.

At this point, the layout of every cycle of loops held on the knitting machine bed during each construction step has been determined, as well as their left-to-right order on the bed. However, our system still needs to assign horizontal offsets to each cycle. To do this, our system begins by arranging the cycles as compactly as possible, given the left-to-right order determined by the directed acyclic graph enumeration step above. Starting from this configuration, our system then makes a series of optimal adjustments – where each adjustment involves adding or removing between zero and ten needles of space between two adjacent layouts on every construction step. Once no adjustment that lowers the summed absolute distance between loop positions in subsequent steps is available, the scheduling is finished, and needles are assigned (Figure 7.6e).

At the end of this stage, our system has explicitly computed an *embedding* for the face instructions or fragments of the two-layer augmented stitch mesh.

The next step, is to compute any necessary loop movement achieved by transfer operations to execute the connections described by the embedding. We use the transfer planning algorithm described in McCann et al. [2016] to compute this. Transfer planning involves gener-

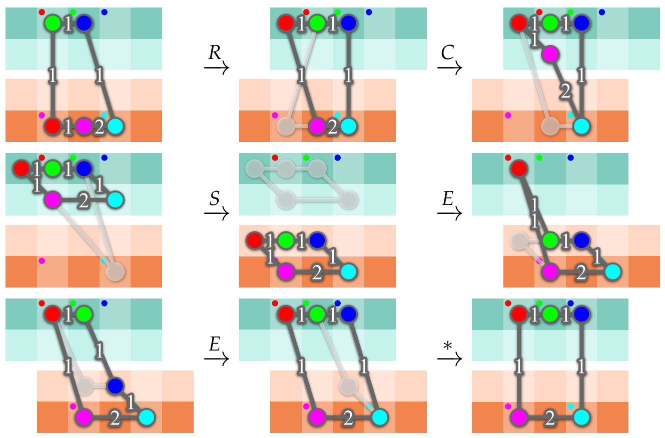


Figure 7.7: A step-by-step transfer plan where target positions of the colored stitches are indicated by the corresponding colored dot. The cycle repeatedly undergoes (R)oll,(C)ollapse, (S)hift and (E)xpand phases until the target state(*) is reached. (Figure from McCann et al. [2016] supplementary video)

ating the necessary transfer operations that moves stitches from one configuration on the needle bed to another. Consider a simple decrease or increase: Since stitches cannot be moved from one needle to another on the same bed, in general, at least some of them have to be 'collapsed' to the opposite bed. This is followed by optionally racking the bed and then stitches are moved to their desired positions. The basic idea of the transfer planning algorithm is to follow a sequence of 'collapse and expand' steps in a way that strictly makes progress towards the target state. McCann et al. [2016] proposes a *roll-goal* penalty function that is the sum of the distance between the current and target position for each stitch along the cycle, and show that it can always be reduced using repeated collapse-expand steps while not introducing any yarn tangles or moving stitches too far apart. Collapse steps may additionally roll stitches to the correct bed when possible. A shift phase may be interleaved to move the collapsed cycle from one bed to another. (see Figure 7.7). Our system interleaves transfer planning between steps to ensure that loops that a face depends on (i.e., incident on incoming edges) are moved to location expected by the face. Face programs that include transfers within their programs are executed independent of the transfer planning as these operations appear as face instructionswithin the instruction graph. This transfer resource planner, does not explicitly track yarn locations and can potentially introduce unintended yarn tangling when a face uses multiple yarns. A full resource planner should incorporate miss instructions to move yarn carriers in a way that does not introduce these tangles. The procedure described above might seem like a reasonable way to generalize for non-planar constraints however this requires correctly establishing and maintaining the topology of the input shape. This is why for the general case, we first establish the topology of the graph by relaxing the metric constraints of slack and then reintroduce them with conservative local rules that do not alter the topology.

7.3 *Code Generation*

The embedded instruction graph provides a straightforward way to generate code, by reading instructions in time order and emitting the appropriate *knitout* code fragment. The resource planning problem described above is performed for slack maintenance, but the same idea of re-ordering instructions can also be used for optimizing schedules.

7.3.1 *Optimizing schedules*

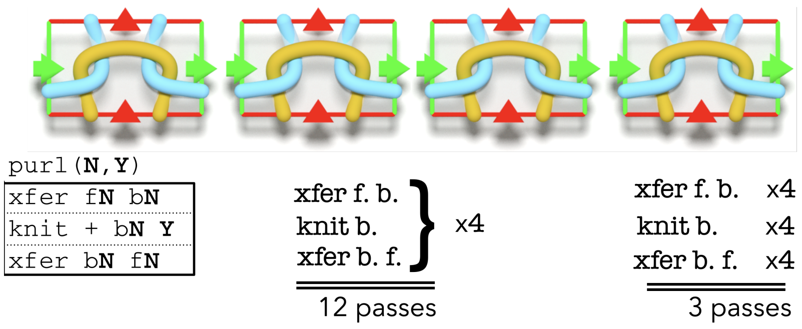


Figure 7.8: The order in which a transfers and (back) knits are constructed for executing a sequence of purl operations can influence the number of passes needed on the machine.

On a single system machine like the one used to fabricate our examples, knit and transfer instructions must be performed in separate carriage movements; this means that a row of N purl stitches – each of which requires an `xfer`, `knit`, `xfer` instruction chain – takes $3N$ carriage movements to fabricate if all the instructions are in operated in sequence but only 3 carriage movements when reordered.

Instead of invoking a single function, the face programs are divided into a preamble, main execution and a postamble that share the same signature:

```
function face_*(dirs, bns, carrier, layer)
```

These three functions allow for coarse instruction re-ordering which becomes important when creating textures with *purl faces* or when scheduling for multiple layers. The operation `layer front` and `layer back` is used to move the loop to the appropriate bed location for the given layer. Consider a purl face (i.e., a back knit) that constructs the stitch on the *opposite* bed and would be defined as:

```
function purl_pre(dirs, bns, carrier, layer){
    layer_front(bns[0])
    xfer bns[0] opposite(bns[0])
}
function purl_main(dirs, bns, carrier, layer){
    knit dirs[0] opposite(bns[0]) carrier
}
function purl_post(dirs, bns, carrier, layer) {
    xfer opposite(bns[0]) bns[0]
}
```

Thus, before generating the final knitout program, instructions are reordered for optimization. A rib pattern is shown in figure 7.9, after reordering instructions the number of passes is significantly reduced.

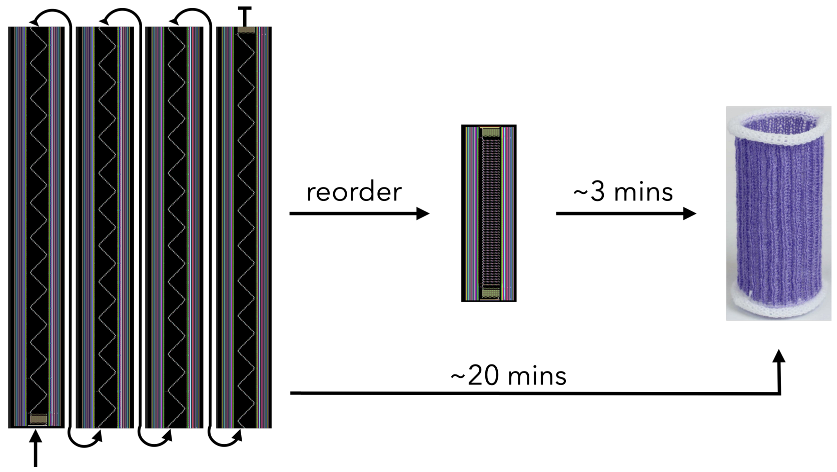


Figure 7.9: Re-ordering instructions to minimize the number of passes in the generated knitout code.

These code movement optimizations also suggest that compiler techniques may be valuable in translating knitout programs (transfer planning sequences, in particular) between different hardware systems while maintaining or improving efficiency – for example, a two-system machine that can knit and transfer in sequence will require a slightly different reordering for an optimal number of passes. For transfers introduced by the purl faces which are heavily used for texturing, such a reordering generates an optimal number of passes. More generally, such re-ordering may fail to identify optimal transfer plans but can significantly lower the number of passes required.

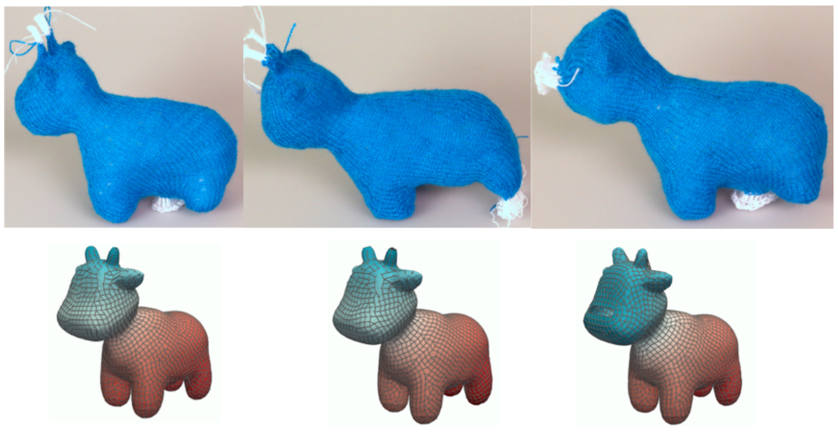
8
Knit Results

I discussed the design space of machine-knittable surface shapes in chapter 4 and the augmented stitch mesh representation to explicitly describe such patterns over a 3D mesh in chapter 5. In chapter 6, I presented multiple techniques to automatically generate knitting patterns from arbitrary 3D meshes. Finally, chapter 7 describes approaches to schedule the augmented stitch mesh and generate code for fabrication. In this final chapter, I present a series of results produced on a Shima Seiki SWGN2 whole-garment knitting machine. Unless otherwise specified, all results have been generated with 2-ply Tamm yarn. For calibrating stitch sizes, a small swatch was knit with the same yarn (as all front-knits) and measured.

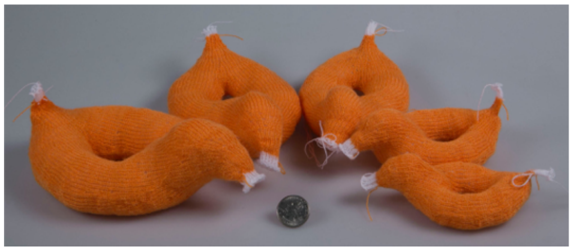
8.1 General two-bed patterns (Incremental approach)

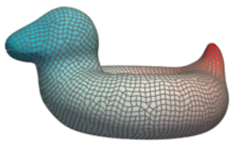
Knit output is inherently stretchy. Identifying the quality of the resulting fit can be difficult. To understand the quality of the incremental remeshing approach, a knit version of the Stanford bunny (a) is stuffed with a foam version of the bunny (b). Concave regions have no reason to adhere to the foam without adhesive but in other regions the output matches the geometry of the input. Although this is fabricated on a *seamless* knitting machine, a distinct seam is visible on the back of the bunny. This is caused by the gap between the beds. To avoid such a visual seam, each row can be slowly rotated such that bed gap is distributed over the body instead of aligned.

The time function used to guide the remeshing approach can influence the quality of the results. Here the same mesh has been remeshed using three different time functions. In the first case, the extrema on the horns ensure they are well captured in the result and the extrema on the belly ensures all four legs are short-rowed in a similar (although not symmetric) manner. In the central shape, the extrema is moved from the belly to the hind legs, although this still captures the shape, the difference in the front and hind limbs are prominent. Finally, in the last version the extrema is moved from the horns to the face, the horns do not protrude very well since the feature size is small.

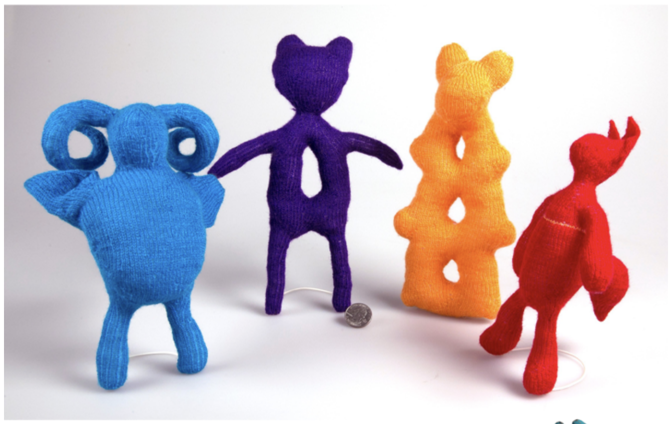


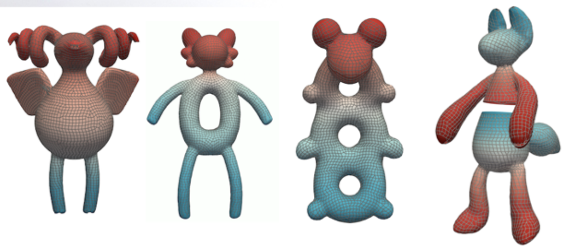
Automatic remeshing allows for easy generation of patterns in multiple scales by simply scaling the input mesh. The resulting patterns however are not constrained in any other way. For more textured patterns, scaling (and other transformations) in a way that retains important characteristics of the pattern would be important.



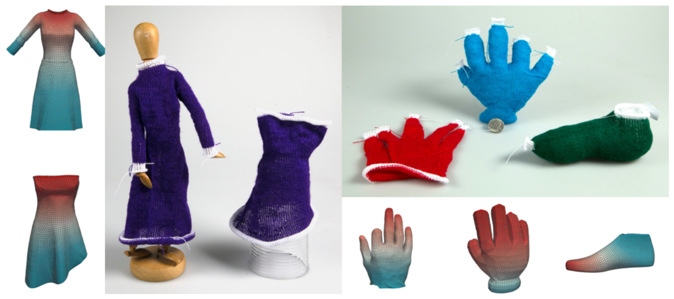


Complex surfaces with non-zero genus can be automatically constructed in one piece. Except for the red fox, these plush objects were all generated in one piece and stuffed after fabrication. Even though our system ensures *balanced* placement of tubes, based on material and number of transfers, splits and merges can fail as in the case of the fox. This was handled by segmenting constructed pattern across a row and sewing up the pieces as a post-process. Figure 8.2 shows how editing can be used to avoid such scenarios instead.

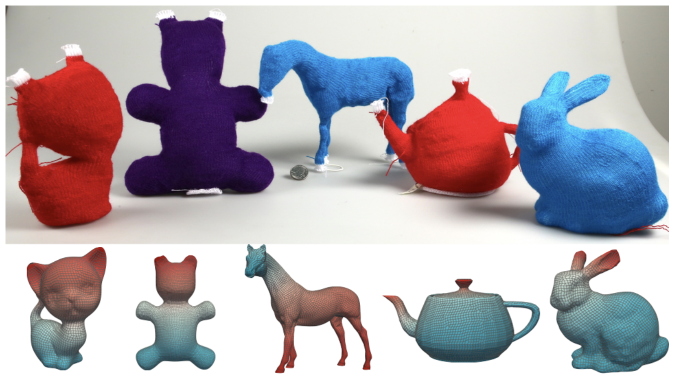




Many commonly knit clothing and accessories can also be generated using meshes as inputs instead of working with flat patterns. The red glove has been constructed from a scanned hand whereas the blue one from a cartoon hand mesh.

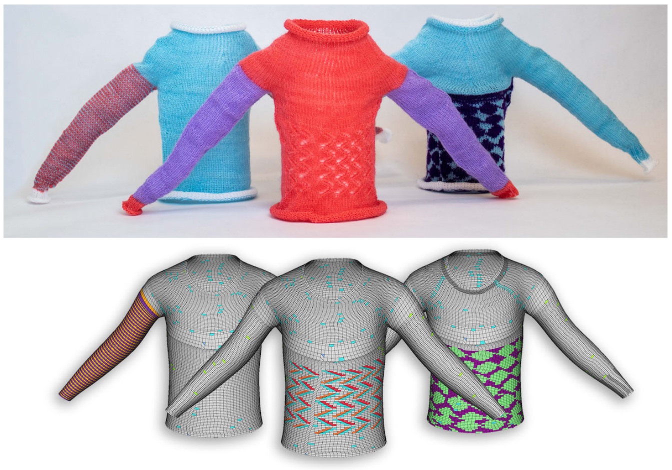


Knit versions of a number of 3D meshes commonly seen in the graphics literature are shown below. Notice that low frequency features in the face of the kitten is smoothed out when knit and stuff. An interesting extension would be to automatically generate pattern or color variants that appropriately highlight these features. The teapot mesh also shows that posing and stuffing (here with clusterfill pillow stuffing) can influence the final shape. Optimizing the input shape to exaggerate regions that would smooth out could lead to better quality outputs.

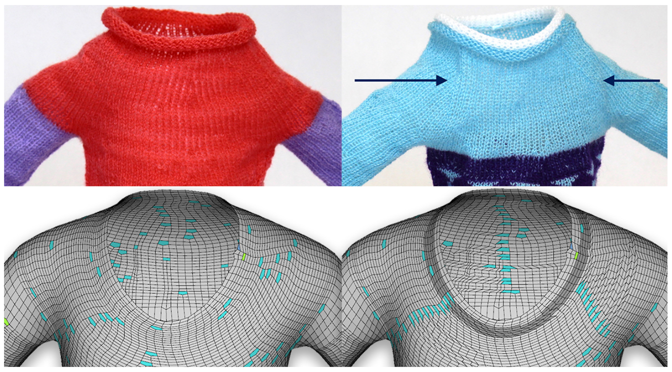


8.2 *General two-bed patterns (hierarchical approach with user editing)*

Using the same sweater mesh as input (generated here with hierarchical remeshing), multiple styles can be rapidly generated to quickly customize and personalize results. Each of these editing sessions took under 15 minutes. Garment patterns carefully constructed by expert designers can be edited by casual users for customization.

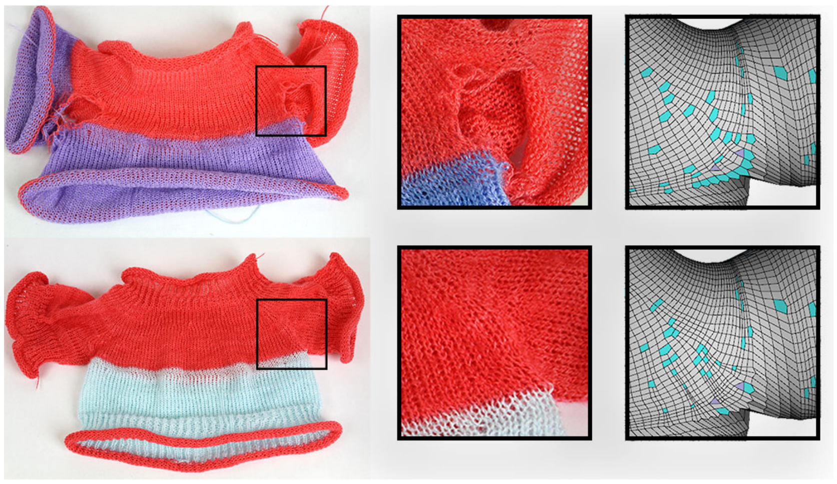


Editing can be used to align decreases and increases for the desired aesthetic effect. The automatic remeshing often distributes increases and decreases to accurately match the 3D geometry but aligned decreases can generate a more pleasing appearance for garments.



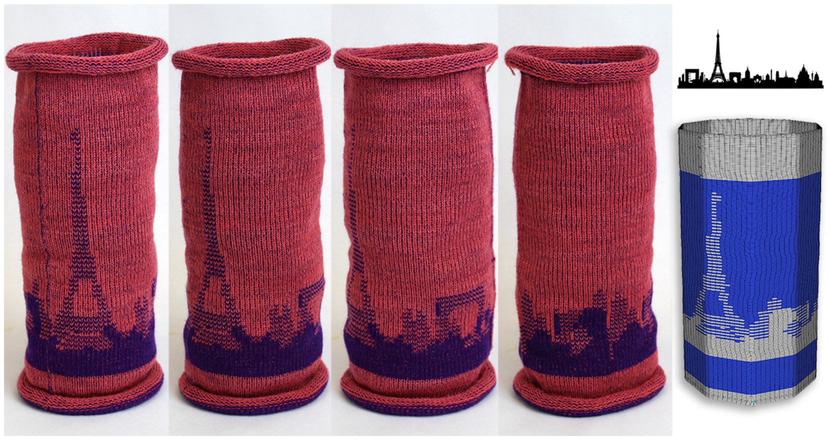
With custom fabrication, a standard pattern like a beanie can be edited to support holes for the ears of a plush toy or create a custom sweaters.

Editing can also be used to avoid high stress by removing shaping and transfers very close to splits and merges. Without editing these operations, this sweater mesh shown above repeatedly failed at the arm joins. Yarn-level simulation is usually used to predict and visualize output shapes. Simulating the construction process itself will be useful in analyzing areas of yarn stress without having to rely on heuristics that limit loop movement.



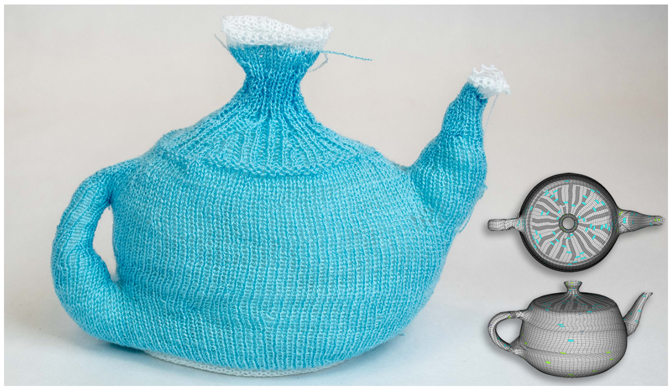
Here, the same input mesh of the Stanford bunny has been edited using augmented stitch mesh face types to generate a ribbed pattern. To account for the stretching of the ribs, the stitch sizes were uniformly edited to fit the bunny appropriately.

Editing can also be driven by an image as input. Here a binary image is used to change the plating style of a face program in the augmented stitch mesh to quickly create a Paris skyline image on the output.



By adding a few yarn breaks and changing the yarn type to a conductive yarn, a regular beanie can be quickly modified to support LEDs.

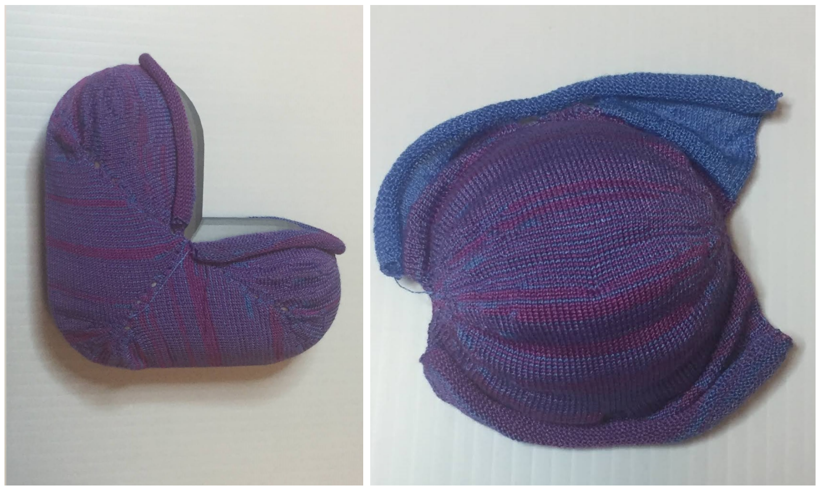
Here a few columns and rows of purl faces have been added to the top of the teapot to influence the final shape.



More examples of fabricated patterns created and edited using the approaches described in this work.

8.3 *Transfer-free patterns*

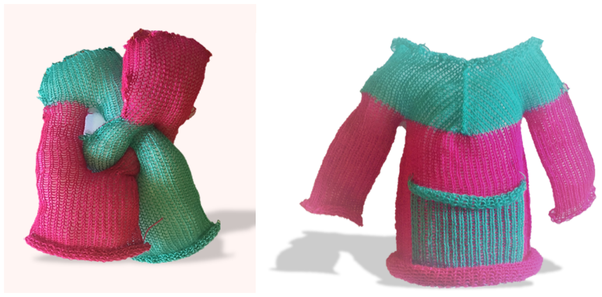
An elbow shape and a hemisphere constructed in 120D Twisted Rayon yarn with only short-rows using the sampling based unfolding strategy. Convex shapes are particularly well suited for this approach since the discretization procedure does not need to deal with partial densities introduced around saddle vertices.



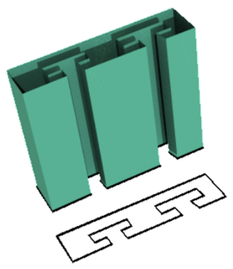
A model of two boxes connected by a narrow sheet unwrapped and patterned without any transfers. The lack of symmetry is expected because of the position of the spine is along one box.

8.4 *Multi-layer patterns with user editing*

A four-layer setup has been used to create this trefoil knot topology. The braiding between tubes is produced on the machine. Many-layer setups require a careful choice of yarn – thinner yarns perform more robustly during the transfers that pass layers around, thicker yarns lessen the apparent loss in resolution. Here we use Yeoman 1-ply supersheen yarn. On the right, a three-layer setup is used with the front-most layer used to add a pocket to this garment.



Here, a four-layer setup is used to extrude a curve that is not balanced. The first three layers used to add pleats to the front side of a skirt, which has more material than the back. Layers provide a natural way to setup such imbalanced curve profiles on the bed.



9
Discussion

In this thesis, we looked at knitting machines as a general programmable machine to produce soft knit objects. To use this machine as a soft 3D printer we looked at coming up with programming abstractions. These abstractions allow high-level 3D shapes to be used as input and programs to construct them can be manipulated in the output (3D) space visually before scheduling them to generate low-level code for fabrication.

This style of programming knitting machines can support a host of novel applications from enabling made-to-measure garments to engineering CAD outputs with carefully programmed materials. There are many ways to improve the pipeline to enable robust production systems.

Representations for Design and Fabrication An important step in digital design for fabrication is the ability to accurately predict designed results (ideally at interactive rates, at least faster than fabrication runtime). Measuring the quality of fabricated results can be challenging since knits are stretchy and yarn can slide. Using techniques from computer vision (and other sensing techniques) to measure output would be important to guarantee quality especially for engineering applications.

Matching a 3D shape is only one of the many design goals an end user or application might have. Ideally, a user might want to specify ideas of fit or comfort for clothing, responses to load or elastic properties for engineering, etc. Design systems must identify ways to represent these objectives and algorithms to optimize for them. Understanding the influence of different material parameters in the context of shaping, behaviour, and, appearance is necessary to model complex, functional objects.

Further, such complex, real-world objects are often made of multiple materials and fabrication techniques. For example, a knit fabric reinforced structure may be cased in concrete or resin. Garments may combine patches of different materials, zippers, 3D printed parts, etc. The augmented stitch mesh representation described in this thesis is machine-agnostic but is restricted to representing a single textiles technique. However, it is a boundary-based representation, encoding connection constraints on the boundary and specialized fabrication routines in the interior. Extending such boundary representations to encode compatibility between different fabrication techniques could generalize this framework further.

DSL design and trade-offs The knitout assembly language used to express local face programs represents atomic operations that the machine can do by assigning an operation type to every cam plate that executes the operation physically. On industrial knitting machines, a sequence of operations may execute in one pass of the carriage and can be viewed as parallel operations. Operations that can be run in parallel would depend on the system – a single system machine can run multiple operations of the same type (all knits or all transfers) in parallel if they match the direction of the carriage.

One natural way to break down a knitting program (pattern) into blocks that can be meaningfully rearranged is to consider pass-level blocks. Basic pass-blocks can be viewed as a design language, building up larger patterns. Nader et al. [2021] propose a similar pass-level setup. Scheduling such the pass-blocks is more natural since each logical pass has only a translational degree of freedom on the machine. Patterns with multiple textures are often not cleanly differentiated by rows in the output, but patches that connect to adjacent ones like jigsaw puzzles. However, these are still constructed row-wise, so each block cannot be constructed entirely together. This makes the scheduling of such blocks more involved, although each block can be subdivided into local passes. The face-programs associated with the augmented stitch mesh provide a domain-specific language that can be viewed as re-entrant routines that execute knitout operations using resource inputs from function arguments. The face-programs described in this chapter are restricted to remain in one layer. Therefore, they cannot describe a volumetric structure easily. A DSL that utilizes the layer-based setup with volumetric units will enable generating interesting output structures like spacer fabrics. One way to generalize face-programs into volumetric structures is to represent layers explicitly in the code and have a different face program for different possible blocks. This is a natural extension of the face programs.

The approaches described above are all similar in how they describe the pattern – explicitly with knitout (or a similar low-level assembly) code. This makes code generation for the entire pattern straightforward. A much more high-level approach is to parametrize patches based on functional properties (for e.g, dimensions, stretchiness, softness, images, heightfields) and material properties (yarn diameter, stiffness, bending, elasticity) and automatically generate code that matches these specifications based on data and simulation.Vidimče et al. [2013] and Chen et al. [2013] have explored such specifications for 3D printing and supporting such a setup for knitting would enable exploring the output space of knit structures in richer ways.

Optimizing fabrication programs Finally, similar to programming in general, applying compiler optimization techniques to fabrication programs can enable targetting the same program for different fabrication hardware. In our code generation pipeline, code re-ordering is used for optimizing program length. Apart from run-time efficiency, fabrication problems also require run-time robustness especially for one-off production. For example, reordering code such that loops are not held for a long time on the machine improves reliability in general. Often, reliability may be more important than optimizing the overall run-time. It would be interesting to profile the machine behavior (for different yarns, speed settings, etc.,) and optimize programs independently to enable design systems to be agnostic to variations in materials and machine settings. On-the-fly program correction by sensing a run-time failure and reactively updating the pattern could further guarantee robustness.

Apart from optimization for production, it is important to consider material wastage and energy efficiency. Textiles contribute significantly to the world's carbon footprint currently and is estimated to use a quarter of the global carbon budget by 2050 (Rana et al. [2015]). While this estimate includes the cost of growing cotton to shipping t-shirts from across the world, improving technology around fabrics – from materials to manufacturing – can have significant impact in the textiles industry.

With the advent of customized and one-off manufacturing services (such as Shapeways, Fast Radius, Xometry, and many more), easy to learn and use design tools (such as OnShape, Fusion360) and support for online marketplaces (Etsy, Amazon, Ebay and so on) the way we design, manufacture and consume is rapidly evolving. Manufacturing techniques need to adapt to these changes and leverage these ecosystems. The ideas put forward in this thesis is a step in the direction of enabling on-demand machine knitting for both designers and manufacturers to produce functional objects using computational design systems.

Bibliography

Adidas. Adidas knit for you. [Online]. Available from: http://adidasknitforyou.com/, 2019.

Pankaj K Agarwal, Boris Aronov, Joseph O'Rourke, and Catherine A Schevon. Star unfolding of a polytope with applications. *SIAM Journal on Computing*, 26(6):1689–1713, 1997.

Ergun Akleman, Jianer Chen, Qing Xing, and Jonathan L Gross. Cyclic plain-weaving on polygonal mesh surfaces with graph rotation systems. *ACM Transactions on Graphics (TOG)*, 28(3):1–8, 2009.

Lea Albaugh, Scott Hudson, and Lining Yao. Digital fabrication of soft actuated objects by machine knitting. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, New York, NY, USA, 2019. Association for Computing Machinery.

Lea Albaugh, James McCann, Scott E. Hudson, and Lining Yao. *Engineering Multifunctional Spacer Fabrics Through Machine Knitting*. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450380966. URL https://doi.org/10.1145/3411764.3445564.

Carlos Aliaga, Carlos Castillo, Diego Gutierrez, Miguel A. Otaduy, Jorge Lopez-Moreno, and Adrian Jarabo. An appearance model for textile fibers. *Computer Graphics Forum*, 36(4):35–45, 2017.

AutoCAD. Autocad. https://www.autodesk.com/products/autocad/.

Shai Avidan and Ariel Shamir. Seam carving for content-aware image resizing. In *ACM SIGGRAPH 2007 papers*, pages 10–es. 2007.

Aric Bartle, Alla Sheffer, Vladimir G. Kim, Danny M. Kaufman, Nicholas Vining, and Floraine Berthouzoz. Physics-driven pattern adjustment for direct 3d garment editing. *ACM Trans. Graph.*, 35(4): 1–11, Jul 2016.

Chelsea Battell. Domain specific language for modular knitting pattern definitions: Purl. *ArXiv*, abs/1606.08708, 2016.

Sarah-Marie Belcastro. Every topological surface can be knit: A proof. *Journal of Mathematics and the Arts*, 3(2):67–83, 2009.

Donald J Berndt and James Clifford. Using dynamic time warping to find patterns in time series. In *KDD workshop*, volume 10, pages 359–370. Seattle, WA, 1994.

Floraine Berthouzoz, Akash Garg, Danny M Kaufman, Eitan Grinspun, and Maneesh Agrawala. Parsing sewing patterns into 3d garments. *ACM Trans. Graph. (TOG)*, 32(4):85, 2013.

David Bommes and Leif Kobbelt. Accurate computation of geodesic distance fields for polygonal curves on triangle meshes. In *VMV*, volume 7, pages 151–160, 2007.

David Bommes, Bruno Lévy, Nico Pietroni, Enrico Puppo, Claudio Silva, Marco Tarini, and Denis Zorin. Quad-mesh generation and processing: A survey. In *Computer Graphics Forum*, volume 32, pages 51–76, 2013.

Sofien Bouaziz, Sebastian Martin, Tiantian Liu, Ladislav Kavan, and Mark Pauly. Projective dynamics: Fusing constraint projections for fast simulation. *ACM Trans. Graph.*, 33(4):154:1–154:11, July 2014.

Marcel Campen, David Bommes, and Leif Kobbelt. Dual loops meshing: quality quad layouts on manifolds. *ACM Transactions on Graphics (TOG)*, 31(4):1–11, 2012.

Michel Carignan, Ying Yang, Nadia Magnenat Thalmann, and Daniel Thalmann. Dressing animated synthetic actors with complex deformable clothes. *ACM SIGGRAPH'92*, pages 99–104, 1992.

Desai Chen, David IW Levin, Piotr Didyk, Pitchaya Sitthi-Amorn, and Wojciech Matusik. Spec2fab: A reducer-tuner model for translating specifications to 3d prints. *ACM Transactions on Graphics (TOG)*, 32 (4):1–10, 2013.

Yanyun Chen, S. Lin, Hua Zhong, Ying-Qing Xu, Baining Guo, and Heung-Yeung Shum. Realistic rendering and animation of knitwear. *IEEE Transactions on Visualization and Computer Graphics*, 9(1):43–55, Jan 2003.

G. Cirio, J. Lopez-Moreno, and M. A. Otaduy. Yarn-level cloth simulation with sliding persistent contacts. *IEEE Transactions on Visualization and Computer Graphics*, 23(2):1152–1162, Feb 2017.

Gabriel Cirio, Jorge Lopez-Moreno, David Miraut, and Miguel A. Otaduy. Yarn-level simulation of woven cloth. *ACM Trans. Graph.*, 33(6):207:1–207:11, November 2014.

Gabriel Cirio, Jorge Lopez-Moreno, and Miguel A. Otaduy. Efficient simulation of knitted cloth using persistent contacts. In *Proceedings of the 14th ACM SIGGRAPH / Eurographics Symposium on Computer Animation*, SCA '15, pages 55–61. ACM, 2015.

Keenan Crane, Clarisse Weischedel, and Max Wardetzky. Geodesics in Heat: A New Approach to Computing Distance Based on Heat Flow. *ACM Trans. Graph.*, 32, 2013.

Tory M Cross and Daniel A Podhajny. Knit article of footwear with customized midsole and customized cleat arrangement, November 21 2017.

Ultimaker Cura. Advanced 3d printing software, made accessible. https://www. ultimaker. com/en/products/ultimaker-cura-software, 2018.

Therese De Dillmont. *Encyclopedia of Needlework*. Th. de Dillmont, 1900.

Phillipe Decaudin, Dan Julius, Jamie Wither, Laurence Boissieux, Alla Sheffer, and Marie-Paule Cani. Virtual garments: A fully geometric approach for clothing design. *CG Forum (Eurographics)*, 25(3):625–634, 2006.

Erik D Demaine and Anna Lubiw. A generalization of the source unfolding of convex polyhedra. In *Spanish Meeting on Computational Geometry*, pages 185–199. Springer, 2011.

Erik D Demaine and Joseph O'Rourke. *Geometric folding algorithms: linkages, origami, polyhedra*. Cambridge university press, 2007.

Erik D Demaine and Joseph O'Rourke. A survey of folding and unfolding in computational geometry. *Combinatorial and computational geometry*, 52:167–211, 2005.

Mario Deuss, Anders Holden Deleuran, Sofien Bouaziz, Bailin Deng, Daniel Piker, and Mark Pauly. *ShapeOp—A Robust and Extensible Geometric Modelling Paradigm*, pages 505–515. Springer International Publishing, Cham, 2015.

Shen Dong, Scott Kircher, and Michael Garland. Harmonic functions for quadrilateral remeshing of arbitrary manifolds. *Computer Aided Geometric Design*, 22(5):392–423, 2005.

Herbert Edelsbrunner and John Harer. *Computational topology: an introduction*. American Mathematical Soc., 2010.

Jonny Farringdon, Andrew J Moore, Nancy Tilbury, James Church, and Pieter D Biemond. Wearable sensor badge and sensor jacket for context awareness. In *Digest of Papers. Third International Symposium on Wearable Computers*, pages 107–113. IEEE, 1999.

Fusion360. Fusion360. https://www.autodesk.com/products/fusion-360/.

Sergei Grishanov, Vadim Meshkov, and Alexander Omelchenko. A topological study of textile structures. part i: An introduction to topological methods. *Textile Research Journal*, 79(8):702–713, 2009a.

Sergei Grishanov, Vadim Meshkov, and Alexander Omelchenko. A topological study of textile structures. part i: An introduction to topological methods. *Textile Research Journal*, 79(8):702–713, 2009b.

E. Groller, R. T. Rau, and W. Strasser. Modeling and visualization of knitwear. *IEEE Transactions on Visualization and Computer Graphics*, 1 (4):302–310, Dec 1995.

Eduard Gröller, René T Rau, and Wolfgang Straßer. Modeling textiles as three dimensional textures. In *Rendering Techniques' 96*, pages 205–214. Springer, 1996.

Ruslan Guseinov, Eder Miguel, and Bernd Bickel. Curveups: Shaping objects from flat plates with tension-actuated curvature. *ACM Trans. Graph.*, 36(4):64:1–64:12, July 2017.

Claire Harvey, Emily Holtzman, Joy Ko, Brooks Hagan, Rundong Wu, Steve Marschner, and David Kessler. Weaving objects: spatial design and functionality of 3d-woven textiles. *Leonardo*, 52(4):381–388, 2019.

Megan Hofmann, Lea Albaugh, Ticha Sethapakadi, Jessica Hodgins, Scott E. Hudson, James McCann, and Jennifer Mankoff. Knitpicking textures: Programming and modifying complex knitted textures for machine and hand knitting. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, UIST '19, pages 5–16, New York, NY, USA, 2019. Association for Computing Machinery.

Donald House and David Breen. *Cloth modeling and animation*. AK Peters/CRC Press, 2000.

Hong Hu, Zhengyue Wang, and Su Liu. Development of auxetic fabrics using flat knitting technology. *Textile Research Journal*, 81(14): 1493–1502, 2011.

Yuanming Hu, Tzu-Mao Li, Luke Anderson, Jonathan Ragan-Kelley, and Frédo Durand. Taichi: A language for high-performance computation on spatially sparse data structures. *ACM Trans. Graph.*, 38 (6), November 2019.

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

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. Knitting a 3d model. volume 27, pages 1737–1743, 2008a.

Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. Knitty: 3d modeling of knitted animals with a production assistant interface. In *Eurographics*, 2008b.

ITMA. Itma textile & garment technology exhibition. [Online]. Available from: `https://www.itma.com`, 2019.

Wenzel Jakob, Adam Arbree, Jonathan T. Moon, Kavita Bala, and Steve Marschner. A radiative transfer framework for rendering materials with anisotropic structure. *ACM Trans. Graph.*, 29(4):53:1–53:13, 2010.

Wenzel Jakob, Marco Tarini, Daniele Panozzo, and Olga Sorkine-Hornung. Instant field-aligned meshes. *ACM Trans. Graph.*, 34(6): 189–1, 2015.

Chenfanfu Jiang, Theodore Gast, and Joseph Teran. Anisotropic elastoplasticity for cloth, knit and hair frictional contact. *ACM Trans. Graph.*, 36(4):152:1–152:14, July 2017.

Ben Jones, Yuxuan Mei, T Gotfrid, Haisen Zhao, Jennifer Mankoff, and Adriana Schulz. Computational design of knit templates. *ACM Transactions on Graphics (to appear)*, 2020.

Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Simulating knitted cloth at the yarn level. *ACM Trans. Graph. (SIGGRAPH'08)*, 27(3):65, 2008.

Levi Kapllani, Chelsea Amanatides, Genevieve Dion, Vadim Shapiro, and David E Breen. Topoknit: A process-oriented representation for modeling the topology of yarns in weft-knitted textiles. *arXiv preprint arXiv:2101.04560*, 2021.

Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. Knitting skeletons: A computer-aided design tool for shaping and patterning of knitted garments. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology*, pages 53–65, 2019a.

Alexandre Kaspar, Tae-Hyun Oh, Liane Makatura, Petr Kellnhofer, and Wojciech Matusik. Neural inverse knitting: From images to manufacturing instructions. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning*

Research, pages 3272–3281, Long Beach, California, USA, 09–15 Jun 2019b. PMLR.

Alexandre Kaspar, Kui Wu, Yiyue Luo, Liane Makatura, and Wojciech Matusik. Knit sketching: from cut & sew patterns to machine-knit garments. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 40(4), 2021.

Pramook Khungurn, Daniel Schroeder, Shuang Zhao, Kavita Bala, and Steve Marschner. Matching real fabrics with micro-appearance models. *ACM Trans. Graph.*, 35(1):1:1–1:26, 2015.

Stephen Kiazyk and Anna Lubiw. Star unfolding from a geodesic curve. *Discrete & Computational Geometry*, 56(4):1018–1036, 2016.

Fredrik Kjolstad, Shoaib Kamil, Jonathan Ragan-Kelley, David I. W. Levin, Shinjiro Sueda, Desai Chen, Etienne Vouga, Danny M. Kaufman, Gurtej Kanwar, Wojciech Matusik, and et al. Simit: A language for physical simulation. *ACM Trans. Graph.*, 35(2), March 2016.

Chelsea Knittel, Diana Nicholas, Reva Street, Caroline Schauer, and Genevieve Dion. Self-folding textiles through manipulation of knit stitch architecture. *Fibers*, 3(4):575–587, Dec 2015.

Mina Konaković, Keenan Crane, Bailin Deng, Sofien Bouaziz, Daniel Piker, and Mark Pauly. Beyond developable: computational design and fabrication with auxetic materials. *ACM Transactions on Graphics (TOG)*, 35(4):1–11, 2016.

GAV Leaf. 4—models of the plain-knitted loop. *Journal of the Textile Institute Transactions*, 51(2):T49–T58, 1960.

GAV Leaf and A Glaskin. 43—the geometry of a plain knitted loop. *Journal of the Textile Institute Transactions*, 46(9):T587–T605, 1955.

Jonathan Leaf, Rundong Wu, Eston Schweickart, Doug L. James, and Steve Marschner. Interactive design of yarn-level cloth patterns. *ACM Trans. Graph. (Proceedings of SIGGRAPH Asia 2018)*, 37(6), 11 2018.

Bruno Levy. Laplace-beltrami eigenfunctions towards an algorithm that" understands" geometry. In *IEEE International Conference on Shape Modeling and Applications 2006 (SMI'06)*, pages 13–13. IEEE, 2006.

Minchen Li, Alla Sheffer, Eitan Grinspun, and Nicholas Vining. Foldsketch: Enriching garments with physically reproducible folds. *ACM Trans. Graph.*, 37(4), 2018.

Jenny Lin and James McCann. An artin braid group representation of knitting machine state with applications to validation and optimization of fabrication plans. 2021.

Jenny Lin, Vidya Narayanan, and James McCann. Efficient transfer planning for flat knitting. In *Proceedings of the 2Nd ACM Symposium on Computational Fabrication*, SCF '18, pages 1:1–1:7, New York, NY, USA, 2018. ACM.

Hsueh-Ti Derek Liu and Alec Jacobson. Cubic stylization. *arXiv preprint arXiv:1910.02926*, 2019.

Zishun Liu, Xingjian Han, Yuchen Zhang, Xiangjia Chen, Yu-Kun Lai, Eugeni L Doubrovski, Emily Whiting, and Charlie CL Wang. Knitting 4d garments with elasticity controlled for body motion. *ACM Transactions on Graphics (TOG)*, 40(4):1–16, 2021.

Jorge Lopez-Moreno, David Miraut, Gabriel Cirio, and Miguel A. Otaduy. Sparse GPU voxelization of yarn-level cloth. *Computer Graphics Forum*, pages 1–13, 2015.

Fujun Luan, Shuang Zhao, and Kavita Bala. Fiber-level on-the-fly procedural textiles. In *Computer Graphics Forum*, volume 36, pages 123–135. Wiley Online Library, 2017.

Yiyue Luo, Kui Wu, Tomás Palacios, and Wojciech Matusik. *KnitUI: Fabricating Interactive and Sensing Textiles with Machine Knitting*. Association for Computing Machinery, New York, NY, USA, 2021. ISBN 9781450380966. URL https://doi.org/10.1145/3411764.3445780.

Abhishek Madan, Alec Jacobson, and David IW Levin. Diffusion structures for architectural stripe pattern generation. *arXiv e-prints*, pages arXiv–2011, 2020.

Luigi Malomo, Jesús Pérez, Emmanuel Iarussi, Nico Pietroni, Eder Miguel, Paolo Cignoni, and Bernd Bickel. Flexmaps: Computational design of flat flexible shells for shaping 3d objects. *ACM Trans. on Graphics - Siggraph Asia 2018*, 37(6):14, nov 2018.

Shashank Markande and Elisabetta Matsumoto. A topological perspective on knitted fabrics. In *APS March Meeting Abstracts*, volume 2019 of *APS Meeting Abstracts*, page K63.004, January 2019.

MasterCam. Mastercam. https://www.mastercam.com/.

Ali Maziz, Alessandro Concas, Alexandre Khaldi, Jonas Stålhand, Nils-Krister Persson, and Edwin WH Jager. Knitting and weaving artificial muscles. *Science advances*, 3(1):e1600327, 2017.

James McCann. The "knitout" (.k) file format. [Online]. Available from: https://textiles-lab.github.io/knitout/knitout.html, 2017.

James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. A compiler for 3d machine knitting. *ACM Trans. Graph.*, 35(4):49:1–49:11, July 2016.

Michael Meißner and Bernd Eberhardt. The art of knitted fabrics, realistic & physically based modelling of knitted patterns. In *Computer Graphics Forum*, volume 17, pages 355–362. Wiley Online Library, 1998.

Ministry of Supply. Ministry of supply. https://www.ministryofsupply.com/.

Yuki Mori and Takeo Igarashi. Plushie: An interactive design system for plush toys. *ACM Trans. Graph. (SIGGRAPH'07)*, 26(3):45, 2007.

Georges Nader, Yu Han Quek, Pei Zhi Chia, Oliver Weeger, and Sai-Kit Yeung. Knitkit: A flexible system for machine knitting of customizable textiles. *ACM Transactions on Graphics (TOG)*, 40(4), 2021.

Chandrakana Nandi, Anat Caspi, Dan Grossman, and Zachary Tatlock. Programming language tools and techniques for 3d printing. In *2nd Summit on Advances in Programming Languages (SNAPL 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

Vidya Narayanan, Lea Albaugh, Jessica Hodgins, Stelian Coros, and James McCann. Automatic machine knitting of 3d meshes. *ACM Trans. Graph.*, 37(3):35:1–35:15, August 2018.

Vidya Narayanan, Kui Wu, Cem Yuksel, and James McCann. Visual knitting machine programming. *ACM Transactions on Graphics (TOG)*, 38(4):1–13, 2019.

Robert Osada, Thomas Funkhouser, Bernard Chazelle, and David Dobkin. Shape distributions. *ACM Transactions on Graphics (TOG)*, 21(4):807–832, 2002.

Jifei Ou, Daniel Oran, Don Derek Haddad, Joseph Paradiso, and Hiroshi Ishii. Sensorknit: Architecting textile sensors with machine knitting. *3D Printing and Additive Manufacturing*, 6(1):1–11, 2019a.

Jifei Ou, Daniel Oran, Don Derek Haddad, Joseph Paradiso, and Hiroshi Ishii. Sensorknit: Architecting textile sensors with machine knitting. *3D Printing and Additive Manufacturing*, 6(1):1–11, 2019b.

Frederick Thomas Peirce. 5—the geometry of cloth structure. *Journal of the Textile Institute Transactions*, 28(3):T45–T96, 1937.

Danit Peleg. https://danitpeleg.com/, 2018.

Chi-Han Peng and Peter Wonka. Connectivity editing for quad-dominant meshes. In *Proceedings of the Eleventh Eurographics/ACM-SIGGRAPH Symposium on Geometry Processing*, SGP '13, pages 43–52. Eurographics Association, 2013.

Jesús Pérez, Miguel A Otaduy, and Bernhard Thomaszewski. Computational design and automated fabrication of kirchhoff-plateau surfaces. *ACM Transactions on Graphics (TOG)*, 36(4):1–12, 2017.

M. Popescu. *KnitCrete: Stay-in-place knitted fabric formwork for complex concrete structures*. PhD thesis, 2019.

Mariana Popescu, Matthias Rippmann, Tom Van Mele, and Philippe Block. Automated generation of knit patterns for non-developable surfaces. In De Rycke K. et al., editor, *Humanizing Digital Reality*. Springer, Singapore, 2018.

Virginia Postrel. *The fabric of civilization: how textiles made the world*. Hachette UK, 2020.

Helmut Pottmann, Alexander Schiftner, Pengbo Bo, Heinz Schmiedhofer, Wenping Wang, Niccolo Baldassini, and Johannes Wallner. Freeform surfaces from single curved panels. *ACM Transactions on Graphics (TOG)*, 27(3):1–10, 2008.

Helmut Pottmann, Michael Eigensatz, Amir Vaxman, and Johannes Wallner. Architectural geometry. *Computers & graphics*, 47:145–164, 2015.

Jonathan Ragan-Kelley, Andrew Adams, Sylvain Paris, Marc Levoy, Saman Amarasinghe, and Frédo Durand. Decoupling algorithms from schedules for easy optimization of image processing pipelines. *ACM Trans. Graph.*, 31(4):32:1–32:12, July 2012.

S Ramakrishna. Characterization and modeling of the tensile properties of plain weft-knit fabric-reinforced composites. *Composites Science and Technology*, 57(1):1–22, 1997.

Sohel Rana, Subramani Pichandi, Shabaridharan Karunamoorthy, Amitava Bhattacharyya, Shama Parveen, and Raul Fangueiro. 7 carbon footprint of textile. *Handbook of sustainable apparel production*, page 141, 2015.

Alessandro Ranellucci. Reprap/slic3r and the future of 3d printing. *Canessa, E., Fonda, C., and Zennaro, ed.,"Low-cost 3D Printing for Science, Education, and Sustainable Development*, pages 75–82, 2013.

G. Reeb. Sur les points singuliers d'une forme de Pfaff complètement intégrable ou d'une fonction numérique. *Les Comptes rendus de l'Académie des sciences*, 1946.

Gerard Rubio, Triambak Saxena, and Tom Catling. Kniterate. [Online]. Available from: https://www.kniterate.com, 2017.

C Rudd, M Owen, and V Middleton. Mechanical properties of weft knit glass fibre/polyester laminates. *Composites Science and Technology*, 39(3):261–277, 1990.

Ryan Schmidt and Karan Singh. Meshmixer: an interface for rapid mesh composition. In *ACM SIGGRAPH 2010 Talks*, pages 1–1. 2010.

Shima Seiki. Sds-one apex3. [Online]. Available from: http://www.shimaseiki.com/product/design/sdsone_apex/flat/, 2011.

Shima Seiki. Sds-one apex4. [Online]. Available from: https://www.shimaseiki.com/product/design/, 2019.

Mélina Skouras, Bernhard Thomaszewski, Bernd Bickel, and Markus Gross. Computational design of rubber balloons. In *Computer Graphics Forum*, volume 31, pages 835–844, 2012.

Soft Byte Ltd. Designaknit. [Online]. Available from: https://www.softbyte.co.uk/designaknit.htm, 1999.

SolidWorks. Solidworks. https://www.solidworks.com/.

David J Spencer. *Knitting technology: a comprehensive handbook and practical guide*, volume 16. CRC press, 2001.

Stoll. M1plus pattern software. [Online]. Available from: http://www.stoll.com/stoll_software_solutions_en_4/pattern_software_m1plus/3_1, 2011.

Shinjiro Sueda, Garrett L. Jones, David I. W. Levin, and Dinesh K. Pai. Large-scale dynamic simulation of highly constrained strands. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, pages 39:1–39:10, New York, NY, USA, 2011. ACM.

Vitaly Surazhsky, Tatiana Surazhsky, Danil Kirsanov, Steven J Gortler, and Hugues Hoppe. Fast exact and approximate geodesics on meshes. *ACM transactions on graphics (TOG)*, 24(3):553–560, 2005.

Joanne Turney. *The culture of knitting*. Berg, 2009.

Emmanuel Turquin, Jamie Wither, Laurence Boissieux, Marie-Paule Cani, and John Hughes. A sketch-based interface for clothing virtual characters. *IEEE Comp. Graph. and Applications*, 27(1):72–81, 2007.

Nobuyuki Umetani, Danny M. Kaufman, Takeo Igarashi, and Eitan Grinspun. Sensitive couture for interactive garment editing and modeling. *ACM Trans. Graph. (SIGGRAPH'11)*, 30(4):90, 2011.

Jenny Underwood. *The design of 3D shape knitted preforms*. PhD thesis, Fashion and Textiles, RMIT University, 2009.

Unmade. The Unmaking Process.

Josh Vekhter, Jiacheng Zhuo, Luisa F Gil Fandino, Qixing Huang, and Etienne Vouga. Weaving geodesic foliations. *ACM Transactions on Graphics (TOG)*, 38(4):1–22, 2019.

Kiril Vidimče, Szu-Po Wang, Jonathan Ragan-Kelley, and Wojciech Matusik. Openfab: A programmable pipeline for multi-material fabrication. *ACM Trans. Graph.*, 32(4):136:1–136:12, July 2013.

Pascal Volino and Nadia Magnenat-Thalmann. *Virtual Clothing: Theory and Practice*. Springer, 2000.

Pascal Volino, Nadia Magnenat-Thalmann, and Francois Faure. A simple approach to nonlinear tensile stiffness for accurate cloth simulation. *ACM Trans. Graph.*, 28(4):105, 2009.

Paras Wadekar, Prateek Goel, Chelsea Amanatides, Genevieve Dion, Randall D Kamien, and David E Breen. Geometric modeling of knitted fabrics using helicoid scaffolds. *Journal of Engineered Fibers and Fabrics*, 15:1558925020913871, 2020.

Hao Wang. Proving theorems by pattern recognition II. *Bell System Technical Journal*, 40:1–42, 1961.

Huamin Wang. Rule-free sewing pattern adjustment with precision and efficiency. *ACM Trans. Graph.*, 37(4):53:1–53:13, July 2018.

Tuanfeng Y. Wang, Duygu Ceylan, Jovan Popović, and Niloy J. Mitra. Learning a shared shape space for multimodal garment design. volume 37, pages 203:1–203:13, New York, NY, USA, December 2018. ACM.

Kui Wu and Cem Yuksel. Real-time cloth rendering with fiber-level detail. *IEEE Transactions on Visualization and Computer Graphics*, PP (99):1–1, 2017a.

Kui Wu and Cem Yuksel. Real-time fiber-level cloth rendering. In *ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (I3D 2017)*, New York, NY, USA, 2017b. ACM.

Kui Wu, Xifeng Gao, Zachary Ferguson, Daniele Panozzo, and Cem Yuksel. Stitch meshing. *ACM Trans. Graph. (Proceedings of SIGGRAPH 2018)*, 37(4):130:1–130:14, jul 2018.

Kui Wu, Marco Tarini, Cem Yuksel, James Mccann, and Xifeng Gao. Wearable 3d machine knitting: Automatic generation of shaped knit sheets to cover real-world objects. *IEEE Transactions on Visualization and Computer Graphics*, 2021.

Ying-Qing Xu, Yanyun Chen, Stephen Lin, Hua Zhong, Enhua Wu, Baining Guo, and Heung-Yeung Shum. Photorealistic rendering of knitwear using the lumislice. In *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '01, pages 391–398, New York, NY, USA, 2001. ACM.

Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. Stitch meshes for modeling knitted clothing with yarn-level detail. *ACM Trans. Graph. (Proceedings of SIGGRAPH 2012)*, 31(3):37:1–37:12, 2012.

Xiaohui Zhang and Pibo Ma. Application of knitting structure textiles in medical areas. *Autex Research Journal*, 18(2):181–191, 2018.

Shuang Zhao, Wenzel Jakob, Steve Marschner, and Kavita Bala. Building volumetric appearance models of fabric using micro ct imaging. *ACM Trans. Graph.*, 30(4):44:1–44:10, 2011.

Shuang Zhao, Fujun Luan, and Kavita Bala. Fitting procedural yarn models for realistic cloth rendering. *ACM Trans. Graph.*, 35(4):51:1–51:11, 2016a.

Shuang Zhao, Lifan Wu, Frédo Durand, and Ravi Ramamoorthi. Downsampling scattering parameters for rendering anisotropic media. *ACM Trans. Graph.*, 35(6), 2016b.