What's in a cable? Abstracting Knitting Design Elements with Blended Raster/Vector Primitives

Hannah Twigg-Smith
htwigg@uw.edu
University of Washington
Seattle, Washington, USA

Yuecheng Peng
ychpeng@uw.edu
University of Washington
Seattle, Washington, USA

Emily Whiting
whiting@bu.edu
Boston University
Boston, Massachusetts, USA

Nadya Peek
nadya@uw.edu
University of Washington
Seattle, Washington, USA

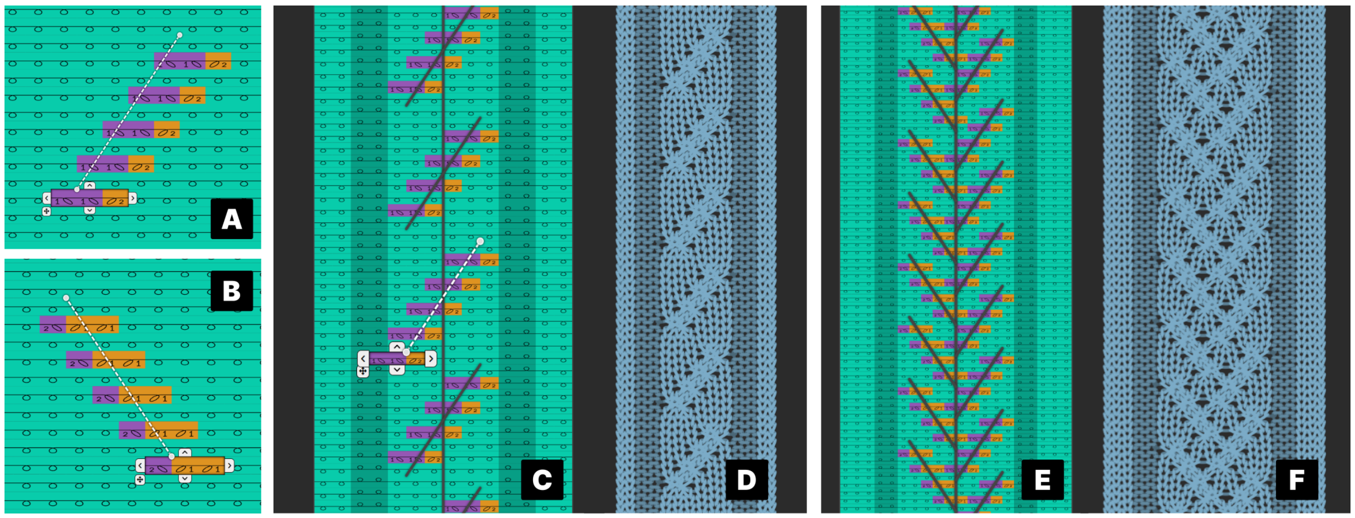

Figure 1: In our design environment for editing knitting charts with blended primitives, a chart of knitting instructions is rasterized from layered vector boundaries and paths with associated raster stitch and yarn blocks. These cabled twist and braid patterns are designed using stitch paths we defined to encode a "right-leaning twist" (A) and a "left-leaning twist" (B). By placing the right twist along a stitch path that defines a purl border (C), we can produce a twist texture (D). By staggering both the left and right twists (E), we can produce a braid texture (F).

ABSTRACT

In chart-based programming environments for machine knitting, patterns are specified at a low level by placing operations on a grid. This highly manual workflow makes it challenging to iterate on design elements such as cables, colorwork, and texture. While vector-based abstractions for knitting design elements may facilitate higher-level manipulation, they often include interdependencies which require stitch-level reconciliation. To address this, we contribute a new way of specifying knits with blended vector and raster primitives. Our abstraction supports the design of interdependent elements like colorwork and texture. We have implemented our blended raster/vector specification in a direct manipulation design tool where primitives are layered and rasterized, allowing for simulation of the resulting knit structure and generation of machine instructions. Through examples, we show how our approach enables higher-level manipulation of various knitting techniques, including intarsia colorwork, short rows, and cables. Specifically, we show how our tool supports the design of complex patterns including origami pleat patterns and capacitive sensor patches.

UIST '24, October 13–16, 2024, Pittsburgh, PA, USA
© 2024 Copyright held by the owner/author(s).
ACM ISBN 979-8-4007-0628-8/24/10
https://doi.org/10.1145/3654777.3676351

CCS CONCEPTS

• **Human-centered computing** → **Interactive systems and tools**; • **Software and its engineering** → **Integrated and visual development environments**.

KEYWORDS

Knitting Patterns, Machine Knitting, Design Tools, Abstractions, Yarn Simulation

ACM Reference Format:
Hannah Twigg-Smith, Yuecheng Peng, Emily Whiting, and Nadya Peek. 2024. What's in a cable? Abstracting Knitting Design Elements with Blended Raster/Vector Primitives. In *The 37th Annual ACM Symposium on User Interface Software and Technology (UIST '24), October 13–16, 2024, Pittsburgh, PA, USA.* ACM, New York, NY, USA, 20 pages. https://doi.org/10.1145/3654777.3676351

1 INTRODUCTION

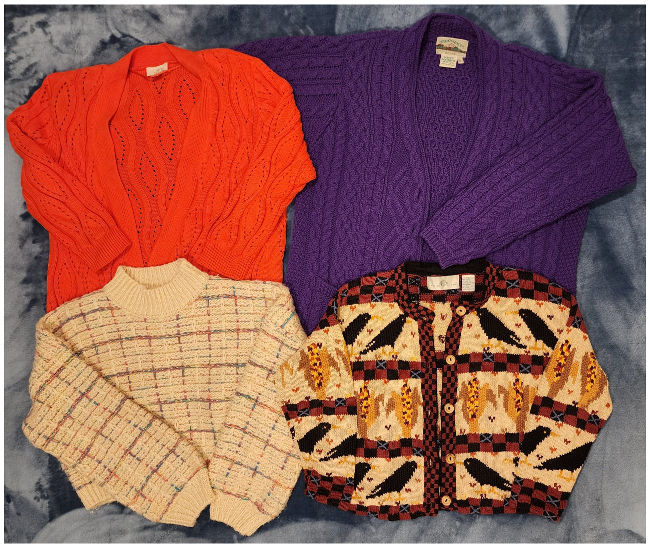
Figure 2: Examples of knit surface patterning effects on a selection of sweaters. As opposed to overall fabric topology (e.g., a sleeve) or the individual stitch, these *mid-level effects* [2] determine the texture and color of knits.

Knitting enables the fabrication of soft objects through a set of discrete operations on a yarn or set of yarns [60]. By combining different stitches and yarns, we can manufacture complex, multi-material, three-dimensional shapes, with varying micro and macro material behaviors [3, 33, 36, 41]. Color patterning (also known as colorwork) and texture effects such as lace, cabling, and ribbing are extensively used in the design of knitwear and can introduce aesthetic and functional attributes to knits [2, 64], including the integration of elements of great interest to HCI researchers such as sensors and shape-changing components [14, 42, 43, 51]. Knitwear designers have common strategies for producing these attributes, but need to ultimately specify their knits with low-level descriptions. The lack of appropriate design representations can result in design challenges, as changing patterns stitch-by-stitch is labor intensive.

For example, a *cable* is a surface texture which looks like a raised column of interwoven or twisted fabric [37]. Cables are a core element of Aran sweaters (such as the purple sweater in Figure 2), where many cables adorn on the surface of a knit to create intricate patterns. Cable knitting involves changing the order of certain loops at regular intervals. A designer has many variables to play with when designing cable structures, including the number of loops at each crossing, the crossing direction, crossing frequency, the column lean, etc., and can additionally incorporate colorwork and texture techniques to produce stunning surface patterns [19]. All of these variables can have a dramatic impact on cable appearance, yet their interdependence defies top-down abstraction: to design new cable structures, you must specify them on the individual stitch level, always reconciling cable-level design decisions with other factors such as shaping.

The problem of design representations in knitting is familiar to HCI and graphics researchers, who have formalized low-level knitting operations in order to develop higher-level primitives that can be used to design new knit objects [45]. This body of prior work has largely focused on 3D shaping-oriented primitives which represent the overall surface topology of knit objects [28, 34, 35, 45, 49, 50]. In contrast, the work presented here seeks to support design tasks which require a different level of abstraction than overall surface topology, namely the design of knit surface patterning effects (i.e., *mid-level effects* [2], examples of which are shown in Figure 2). What kinds of representations might we need to design new cables, lace, colorwork, and texture patterns?

As another example, *intarsia* is a colorwork technique commonly used for patterns with large blocks of color (such as the corn and crow sweater in Figure 2). While it may appear that intarsia patterns would be relatively easy to design in, say, a pixel-based paint tool, there are knitting constraints that a designer must consider during the design of an intarsia pattern, including how yarns of different colors are joined (to prevent gaps between yarns), the direction each yarn is moving in a row, the slope of the boundary between yarns (a slope that is too gentle may lead to long floats), and on which side the yarn starts and ends. An example of the impact of these constraints is provided in Section 4.3. Designing intarsia patterns often involves resolving these constraints via decisions which 1) must happen at a low level, and 2) impact the final appearance of the pattern, sometimes dramatically. For example, this may include something as simple as resizing the height of a motif so that it totals an even number of rows, ensuring a yarn will return to the same side it started on.

As a final motivating example, consider a designer who wants to create different amounts of stretch in different regions of a garment, for example: a sweater with conformal cuffs, a warming densely knit body, an airier knit for the sleeves, and tough patches for the elbows. How can they prototype these different textures? While some textures are well-established, such as a ribbing for cuffs, we have only scratched the surface of the mid-level material effects that are possible. Recent work has explored how mid-level effects can, for example, be tailored to produce material properties ideal for particular regions of garments [41]. How different stitches and yarns impact the final textures in knits is difficult to understand based on low-level descriptions alone, and rapid iteration on prototypes will be crucial as we look to understand the possible material effects of novel functional fibers (e.g., [1, 14]).

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

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

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

In summary, in this paper we contribute:

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

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

2 RELATED WORK

Machine knitting has recently seen a surge of attention from HCI researchers who recognize it as an essential technique for the production of e-textiles, soft interfaces, and functional fabrics, as well as an established outlet for personal expression. In this section we discuss how our contributions fit into the landscape of prior work on knitting design representations and associated tools. Additionally, we review work on rasterization and fabric modeling and simulation which is relevant to the implementation of our design tool. While we focus on machine knitting in this paper, we recognize that interest in textile fabrication also extends to other domains, including weaving [13, 15] and embroidery [17, 18, 20].

2.1 Design Representations for Knit Objects

There appears to be a general consensus among researchers that the complexity of knit programming, and the associated lack of tractable design primitives, poses a barrier to fully exploring the extensive design potential of machine knitting. Prior work has looked to address this challenge, largely through development of high-level shaping primitives which can be used to represent the overall surface topology of knit objects while respecting the fabrication constraints of the knitting process. This includes generalized tubes and sheets [45], which was later extended to include splits and merges and deployed in the context of an interactive design tool [34, 35]. Other representations include stitch meshes [49, 50, 67] and coarse template meshes [28].

We distinguish our work from prior approaches by emphasising our focus on supporting the design of *fabric-level structures* as opposed to high-level 3D shape. High-level shaping primitives largely divorce three-dimensional form from the underlying stitch-level descriptions; this is advantageous for design tasks which are concerned solely with global fabric topology (such as constructing a garment silhouette). However, we argue that the task of designing novel functional and aesthetic fabric-level structures requires fine control over individual loop positions. For example, prior work has demonstrated that small changes to loop positions can have a dramatic impact on the appearance of slip and tuck colorwork patterns [64] and the performance of knit sensors [1, 5]. We therefore think it is crucial to maintain direct access to stitch-level manipulation when designing fabric level effects, which our blended primitives approach supports by enabling this control via raster-based blocks containing stitch-level operations and yarn assignments. Prior work has explored different approaches to surface patterning largely through the development of textual domain-specific programming languages [22, 24, 34]; our work instead presents a visual design environment where operation positions can be controlled via direct manipulation of associated vector paths and boundaries. This is more in line with the approach described by [2], which is specific to brioche knitting.

In parallel with work on design representations, researchers have worked to build out the infrastructure which will be essential for future development of robust machine knitting workflows. This includes process-oriented models of yarn topology [31, 32], domain-specific programming languages [22, 23], compilers [39, 45], algorithms for scheduling and transfer planning [34, 39, 40, 48, 68], and debugging-oriented visualizations [68]. While the work we present in this paper is focused less on infrastructure and more on design representations, we found this prior work to be a valuable reference during implementation of our design tool.

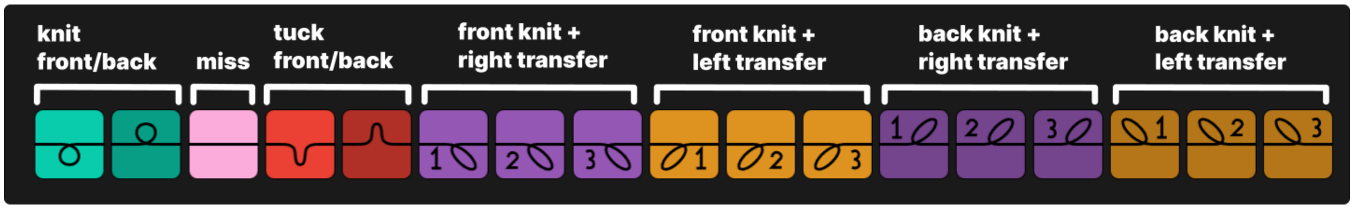
knit front/back
miss
tuck front/back
front knit + right transfer
front knit + left transfer
back knit + right transfer
back knit + left transfer
1
2
3

Figure 3: Knitting operations: The charting language in the blended primitives editing environment supports knit, miss, tuck, and composite transfer operations.

2.2 Fabric Modeling, Simulation, and Rendering

Broadly, yarn simulation has been an area of intense interest in HCI, computer graphics, and textiles communities [11, 29, 30, 38, 69]. Early work achieved draping behavior in knitted fabrics [6], and proposed animation methods with particle systems [47] and splines with animated control points to model the micro-structure of knitted fabrics [57]. Kaldor et al. [29, 30] published the first system capable of simulating entire garments at the yarn level, with computation of contact response of yarn-yarn collisions. Yuksel et al. [69] introduced the stitch mesh representation that incorporated mesh-based and yarn-level relaxation. Further work has focused on efficient implementations through nodal discretizations [53], combined representations to retain large-scale behaviors of cloth [9], and realistic rendering of fabric appearance [10, 70]. Recent work continues to demonstrate new ways in which yarn-level mechanics can improve fabric simulations, such as using implicit contact handling to simulate multi-layer fabric structures [58], animating yarn-level geometry on top of a deforming mesh [61], and inverse-modeling yarn-level mechanics from real-world measurements [62]. For a more extensive review of simulation background we refer to Castillo et al. [10] and Casafranca et al. [9].

A challenge with relying on real-time yarn-level simulation as part of a design workflow is that computational intensity rapidly increases as pieces get larger. Leaf et al. [38] offer yarn-level simulation of swatches at interactive rates, but require GPU implementation. Our system extends the lightweight browser-based simulation introduced in KnitScape [64]. A topology graph [32] is combined with a particle-spring simulation to visualize local deformations in knitted swatches. In our work we demonstrate how mid-level effects like color and texture patterns offer an ideal opportunity to deploy yarn-level simulation in the context of an interactive design tool.

2.3 Rasterization

We draw on foundational concepts in computer graphics for rasterizing geometric primitives and apply them to this domain of knit chart editing. Line drawing algorithms [7, 52, 66] and scanline polygon filling [21] were among the earliest graphics methods for efficiently approximating exact mathematical representations of geometry as pixel positions in the frame buffer. Analogously to rasterization, knitting chart design entails converting geometric lines and regions into discrete stitch positions. Many desired properties are also shared. For example, in origami pleat patterns (Figure 17) the fold lines should be uniformly one stitch wide and avoid artifacts such as gaps when projected onto a knitting chart for consistent folding behavior and appearance.

Pixel art has maintained prominence in computer graphics with popular editors (e.g., Aseprite and Pixelorama), and algorithms to facilitate pixel art creation through image abstraction [16] or rasterizing vector line art with minimal artifacts [26, 27]. Some community-built tools aid in the translation of images to a format readable by manual knitting machines (e.g., img2track [56] and AYAB [4]), but these are focused on translation of existing patterns to a knittable format rather than providing a full design environment. In contrast, our system integrates editor features such as layers and tilemaps into charting-specific workflows.

3 BLENDED PRIMITIVES

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

We propose **blended primitives** for knit pattern design: vector paths and boundaries which can be used to manipulate raster blocks of stitch instructions and yarn assignments. In our design environment, blended primitives are layered and rasterized to create two charts which are overlaid to form a compact chart view: one which encodes yarn assignments and one which encodes stitch instructions. To enable this compact view where multiple yarns can appear in the same row of instructions (as opposed to most editors where only one yarn can appear per row), our system provides automatic yarn separation for fair isle and intarsia patterns, described in Section 6.

Maintaining quick and easy access to low-level stitch and yarn assignments is one of the core goals of blended primitives. In our editor, modifications to the yarn and stitch charts can occur either through direct manipulation of vector elements (e.g. by dragging path segments and control points or changing their stacking order) or via raster edits to their associated stitch and yarn blocks. In

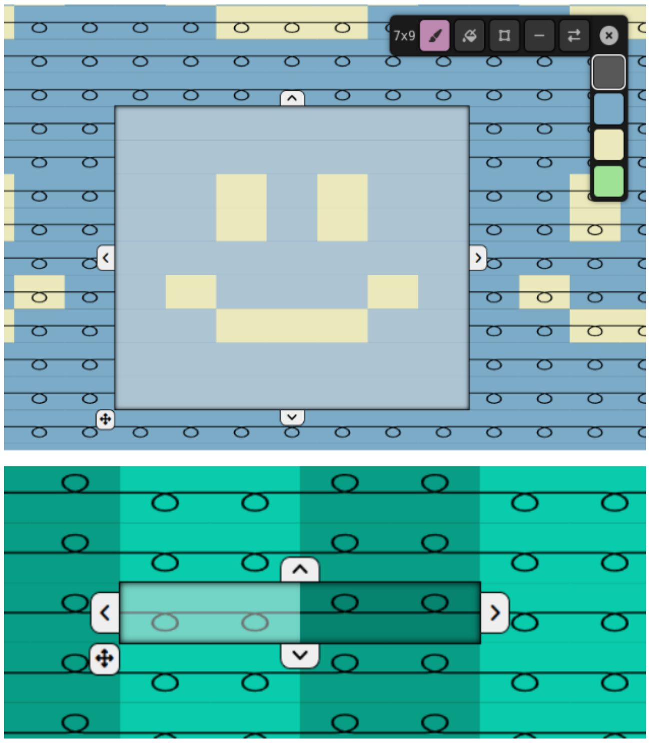

Figure 4: Blocks are raster bitmaps. Yarn blocks (top) encode yarn assignments, and can be edited with yarns from the current yarn palette using pixel editing tools. Similarly, stitch blocks (bottom) can be edited with operations from our supported operation set. Blocks can be quickly resized by dragging the arrows at their edges. The block origin can be easily repositioned using the "move" dragger at the bottom left corner.

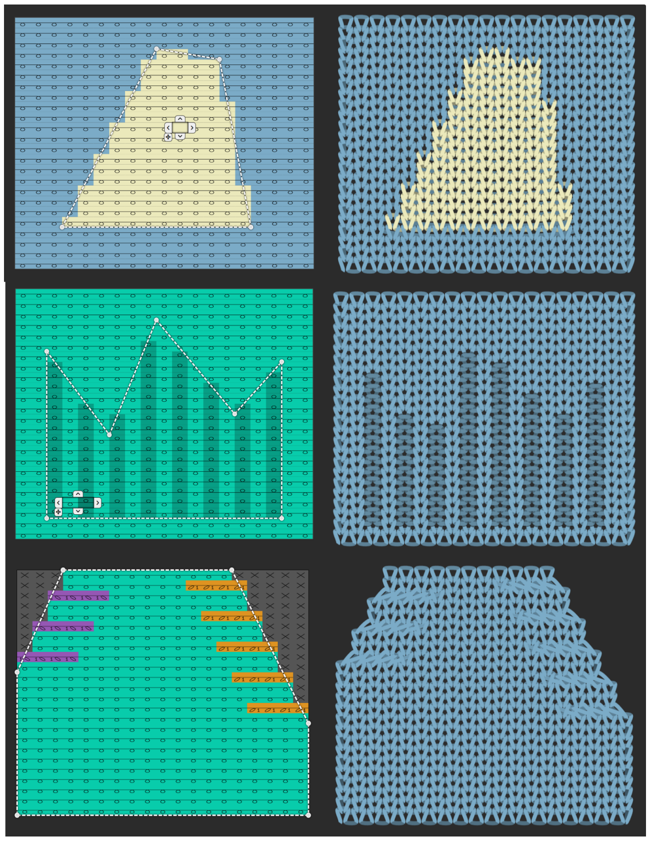

Figure 5: Boundaries can be used to assign yarns (top) and stitch patterns (center) to regions of a chart. They are also used to define the edges of a knit panel, and can optionally have shaping instructions attached to them (bottom).

the rest of this section, we describe the pattern elements used to design knits in our system, give an overview of our tool interface, and provide a walkthrough of an intarsia design workflow in our system. In following sections, we show how blended primitives can be used to design common knitting elements through a series of demonstration examples, including intarsia and fair isle colorwork, lace, and cables. Later, we show two domain demonstrations in which blended primitives are used to design origami-inspired pleat patterns and intarsia-based sensing structures.

3.1 Pattern Elements

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

3.1.1 Blocks. A *block* comprises two bitmaps (one defining yarn assignments and one defining operations) which share a common origin at their lower left corner (Figure 4). All boundaries and paths have an associated block which is used during chart rasterization.. Additionally, "free blocks" can be used as a one-off non-destructive override for yarn and stitch instructions on lower layers, which is useful when designing an individual colorwork or texture motif.

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

Stitch blocks are bitmaps which include supported operations (described in Figure 3) as well as the "transparent" stitch, which inherits the stitch assignment of a lower layer. Yarn blocks are bitmaps which include indices of yarns in the global yarn palette. Similar to stitch blocks, yarn blocks can also include a "transparent" yarn which inherits a yarn assignment from a lower layer. A new block contains a 1x1 bitmap with a transparent stitch and a 1x1 bitmap of a transparent yarn.

3.1.2 Boundaries. A boundary is a closed polyline path which outlines a region in a chart to be filled with a stitch and/or yarn pattern (Figure 5). Figure 12 shows how boundaries can be used to position colorwork motifs and a rib texture for a simple doodle cowl. Boundary edits are not destructive to their stitch and yarn fills, making it easy to use them to lay out and reposition different patterning blocks. This means that patterns like the cowl shown in Figure 12 are very easy to modify, e.g., by dragging the boundary edges. The tiling origin of a stitch and yarn block can be moved

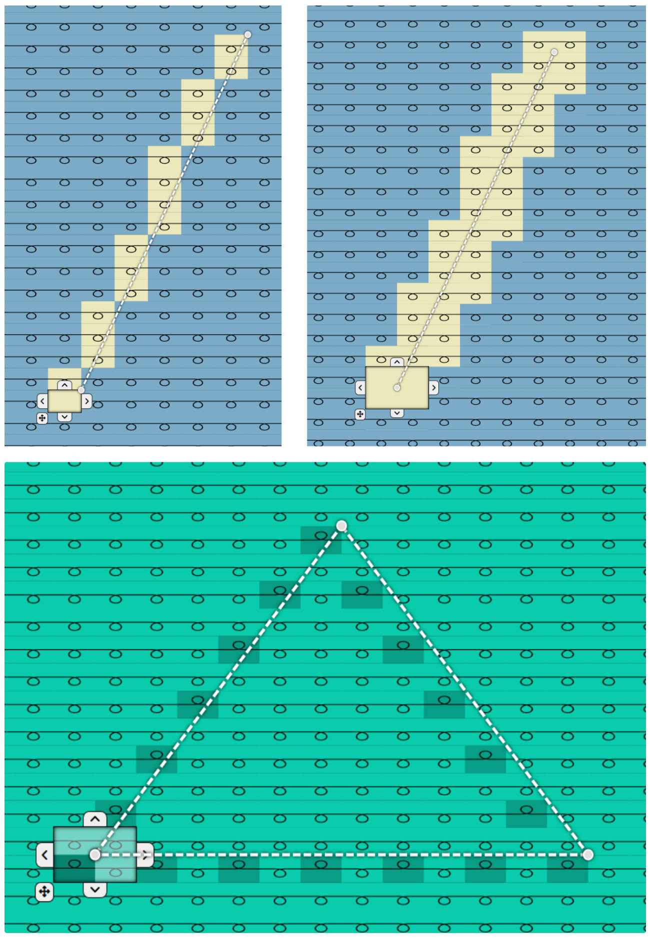

Figure 6: Paths can be used to position stitch and yarn blocks. We can change path appearance by changing the size of the associated block, e.g., to make a single-stitch wide yarn path (top left) into a two-stitch wide yarn path. The same applies for stitch paths (bottom), where we can create a dotted stitch path by adding extra transparent space at the edge of the stitch block.

by dragging the arrows at the bottom left corner of the block. This is used to align blocks relative to their boundaries or other blocks, e.g., to avoid strange repeat cutoffs at the edges or stagger different motifs evenly.

3.1.3 Paths. A path is a polyline along which a stitch and/or yarn block is tiled. Like boundaries, they are easy to manipulate by dragging control points and segments, adding and removing points, and reordering layers. Path appearance is easily changed by editing the associated block (Figure 6). We support four different tiling modes to control block position along the path (described in 6.1.2). The block can also be offset from the associated path using the "move" dragger at its bottom left corner. This can be used, for example, to change which side of a path a single-cell wide block appears on.

3.2 Design Interface

Our design tool (Figure 7) consists of a two-view interface showing our compact chart editor on the left and a yarn-level visualization on the right.

3.2.1 Charting Pane. The charting pane contains a layered view of our overlaid stitch and yarn charts and vector primitives, as well as other interface elements and toolbars which display contextual information. Users manipulate vector elements in the charting pane by dragging on points and segments of vector elements (Figures 5 and 6) and editing their associated blocks (Figure 4, top). Whenever a boundary, path, or block is changed, we rasterize pattern elements to create the underlying stitch and yarn charts (our rasterization process is described further in Section 6). This means that the user never makes direct, destructive edits to a global instruction chart. This is a core advantage to our blended primitives approach: changes to a chart which would be labor-intensive under a traditional pixel-paint workflow (e.g., resizing overall dimensions or translating groups of operations) are instead rapid, incremental, and quickly reversible, recalling foundational principles in the design of interfaces for direct manipulation [59].

The appearance of the chart depends on the active color mode. Each chart cell always contains the symbol of its assigned operation from the stitch chart, but symbols alone can be hard to differentiate (especially when zoomed out), so we offer two color modes for chart editing. In *command mode* each cell is colored according to operations assigned in the stitch chart (shown in Figure 3), and in *yarn mode* each cell is colored according to yarns assigned in the yarn chart. The user can quickly switch between these modes using hotkeys or a button in the bottom toolbar. When the yarn color mode is active, we shade back-bed operations to give them the appearance of depth, which can help give the impression of a rib (an example of this can be seen in Figure 12, B). To enable some visual feedback of yarn color when the command color mode is active, the yarn sequence pane at the far left always shows which yarns are assigned to each row of the chart. The user can also recolor the current yarn palette in this pane, which updates both the chart and yarn visualization. Finally, the bottom toolbar enables the user to switch between three editing modes that control which elements are able to be selected and edited: boundary, path, and block.

3.2.2 Yarn visualization pane. Our interface includes a yarn-level visualization to provide constant feedback on design decisions. The yarn visualization consists of layered yarn segment paths, shaded to give the illusion of depth. It can be viewed from the technical front or back, which can be helpful when inspecting float position on the back of the fabric. The visualization updates, (regenerating yarn topology and redrawing yarn segments), on edits in the charting interface. For smaller patterns (e.g., below 100x100 stitches) these updates are almost instantaneous, but topology generation slows down for larger patterns and those which include multiple yarns and/or short row shaping (and depends on device capability). Therefore we also support a visualization mode where the topology is regenerated on user request, which can be helpful when designing larger or more complex patterns which include multiple yarns. The yarn visualization includes a 2D relaxation simulation where contacts between yarns are modeled as a particle-spring system. The

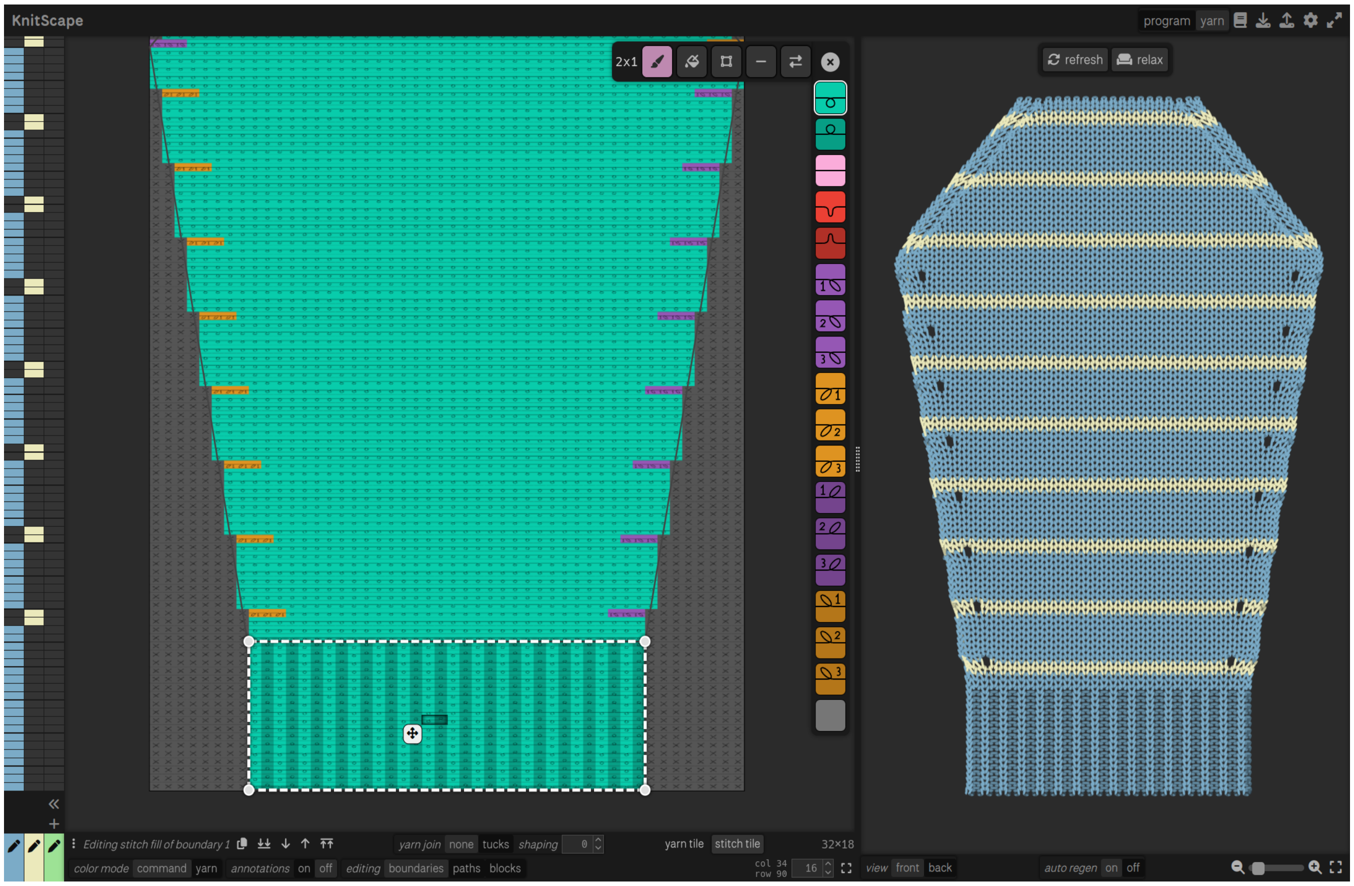

Figure 7: Our design interface consists of a split-pane view of a chart editor and yarn simulation.

relaxation can be run on request via a button in the visualization toolbar, providing a preview of how yarn segments will deform once knit.

3.3 Example Workflow

In this section we describe an example workflow in which we design a simple intarsia pattern, demonstrating how our yarn visualization provides rapid feedback on edits made in our charting interface (Figure 8). On creating a new pattern, the workspace opens in boundary edit mode to show a stockinette swatch: a rectangular boundary with a knit stitch fill and a single yarn fill (Figure 8, A). To begin an intarsia design, the user places control points to define a new boundary (in this case, a circle). By default, all boundaries defined after the first boundary have a transparent stitch and yarn fill, meaning that they will inherit the operations and yarn assignments from prior layers. Therefore, to make the circle a contrasting color, the user assigns a contrasting yarn fill (Figure 8, B). These modifications immediately update the yarn visualization, which now shows the light yellow circle. However, "flipping" the swatch in the visualization pane shows that the blue yarn leaves long floats as it crosses behind the yellow circle (Figure 8, C).

While designing intarsia, these floats can be removed by using another yarn to knit the blue segments which appear to the right of the circle, preventing the yarn from needing to float across the yellow circle. In our editor, this can be accomplished by simply drawing a new boundary that is the same height as the circle and moving it to the layer below the circle (Figure 8, D, left). This removes most of the long blue floats, but two remain as the blue yarn ends up on the wrong side of the pattern due to an odd number of rows. In this example, this is addressed by shifting the circle and the red boundary down by one row, removing the floats (Figure 8, E). If this pattern was knit as-is, there would be gaps where the yarns meet at each row. This is addressed by enabling the "tucks" option on the yarn boundaries to add tucks when two yarns meet at a boundary. Details on this are shown in Figure 9.

4 DEMONSTRATION EXAMPLES

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

4.1 Cables, Braids, and Twists

Cable, braid, and twist textures are produced by selectively crossing groups of stitches to give the appearance of multiple interwoven strands of fabric. Crossed stitches on their own do not have much depth, and cable strands are typically given more definition by using

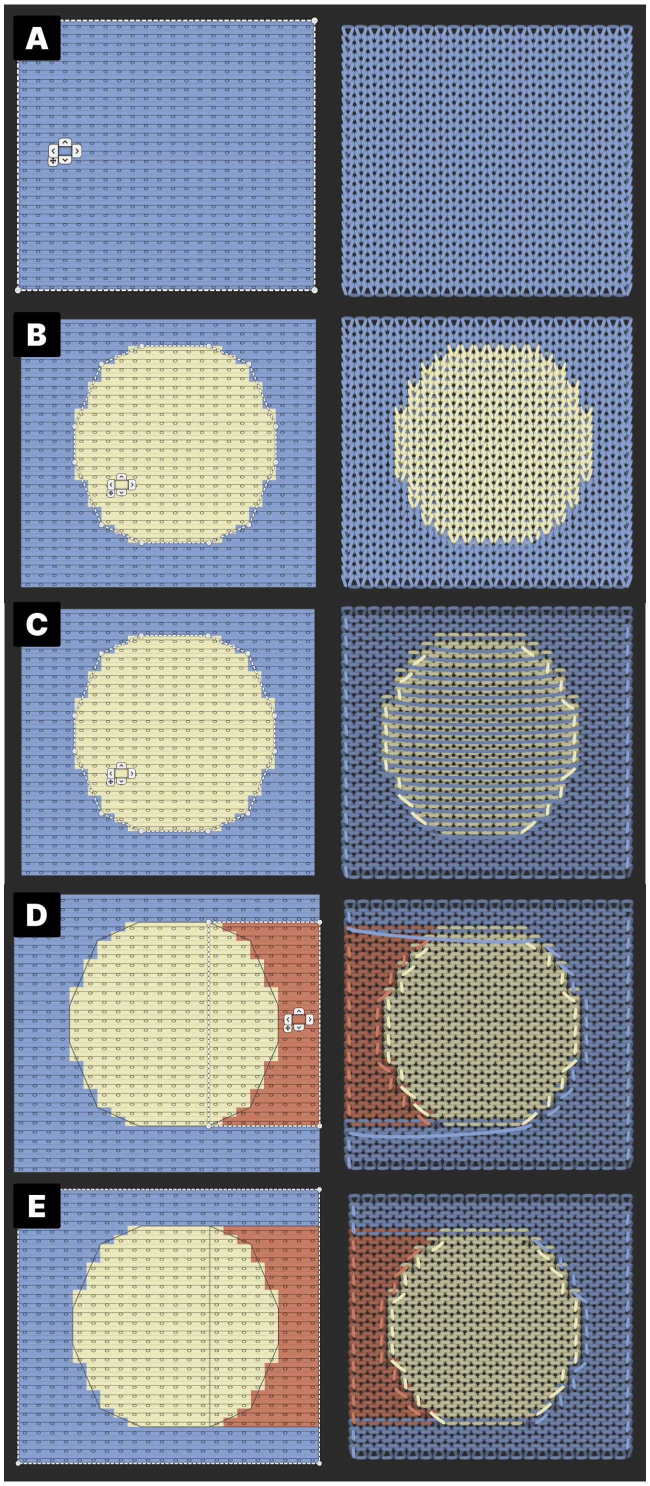

Figure 8: Intarsia knitting: The yarn visualization provides feedback on design decisions made in the charting environment. In this example, it shows the position of yarn floats across the back of the fabric while designing an intarsia pattern.

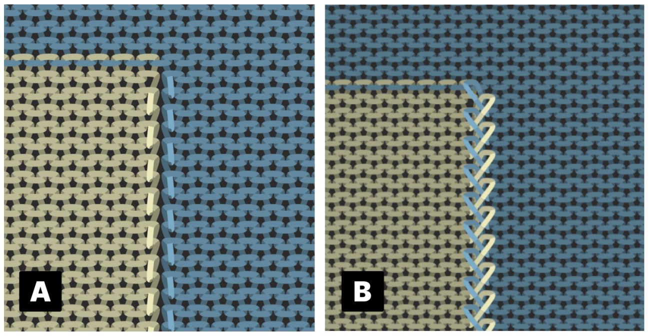

Figure 9: Boundary conditions: The "join" setting controls how yarns are joined at a boundary. When "none" is selected, no change is made. In the case of a vertical join, this will produce a slit (A). The "tucks" option adds a tuck to the previous color in the row, depending on the direction the carriage is moving. These are visible in the simulation view by viewing the back (B).

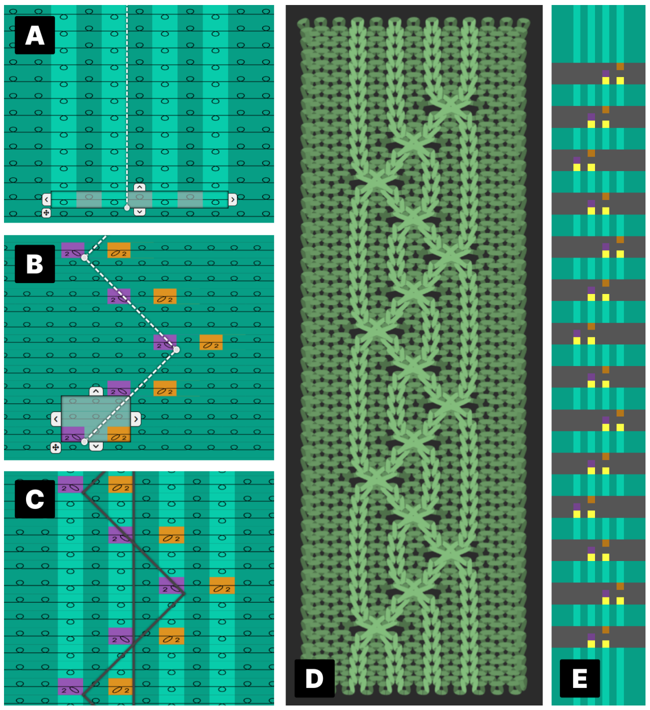

Figure 10: Cable knitting: We designed this cabled lattice by first using a stitch path to make a 1x1 rib (A). A second stitch path is used to place transfers in a zig-zag (B), which we overlay on the rib definition (C). Together, these create a lattice effect which can be seen in the yarn visualization (D) and exported for knitting (E).

knit stitches for the main cable body and purl stitches as a background. There are multiple ways to design cable structures using blended primitives. We found it easiest to control the base knit/purl

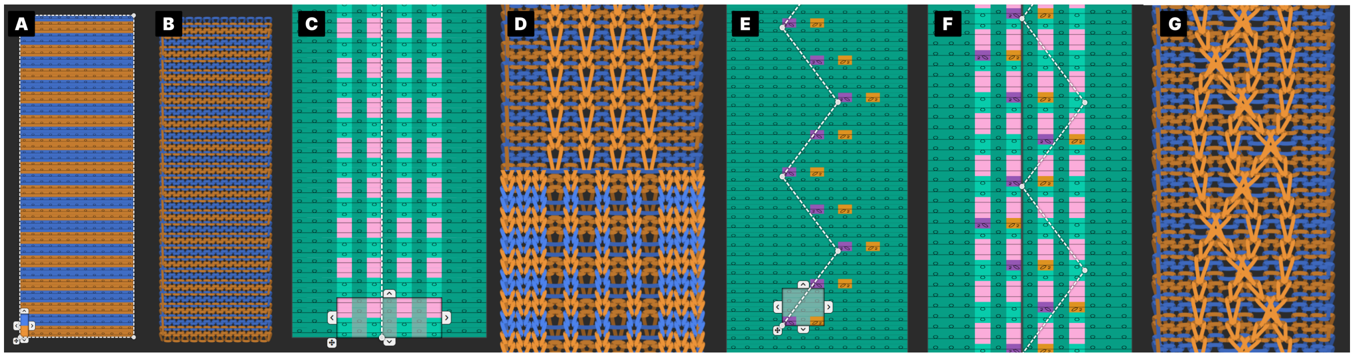

Figure 11: Colorwork cable knitting: To design a colorwork cable lattice, we begin by (A) applying a striped yarn fill and purl stitch fill to a rectangular boundary. A (B) yarn-level visualization of the resulting knit gives constant feedback on design decisions. Switching to the operation view mode, we add a (C) stitch path to overlay a 1x1 rib texture which includes miss stitches every two rows. The (D, top) resulting rib only includes orange yarns, as the blue yarns are (D, bottom) floated across the back. Next, we create (E) a stitch path which tiles transfers in a zigzag pattern. When (F) overlaid on the rib texture, the (G) transfers move loops between the vertical lattice ribs to give them an interwoven effect.

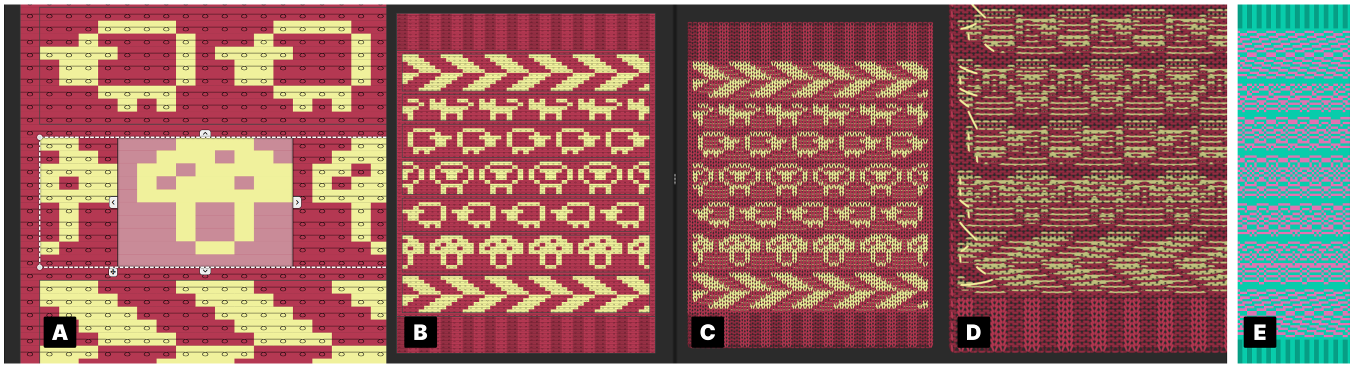

Figure 12: Colorwork repeats: Yarn blocks are tiled to fill their boundaries (A). Boundaries enable easy positioning of colorwork and texture motifs, such as to design this sheep-inspired doodle cowl with ribbed edge (B). The simulation pane shows a yarn-level visualization of the resulting knit (C). Yarn floats can be viewed by flipping the visualization to view the technical back (D). The resulting time-needle view can be seen in (E).

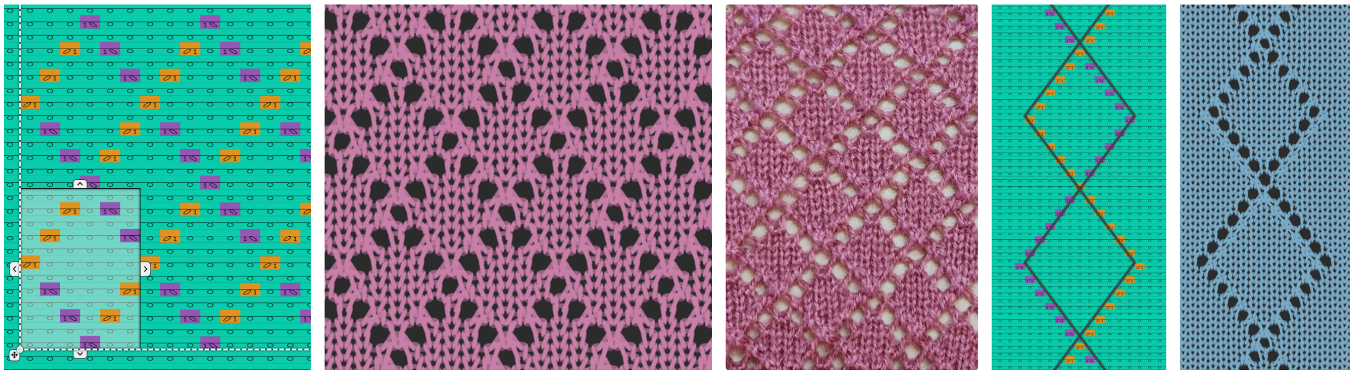

Figure 13: Lace knitting: We support multiple approaches to lace design. Lace patterns can be designed by using a stitch block to fill a boundary (left). This allows the quick application of a repeating lace texture to a region. Stitch paths can also be used to design lace textures (right). This method is well-suited to individual lace motifs.

texture separately from the transfers. We did this by defining separate stitch paths for the knit border and the transfers. Example cable, braid, and twists can be seen in Figures 1 and 10. By overlaying the texture and transfer paths atop a background with a striped yarn fill, we can also design and simulate colorwork cable structures such as the orange and blue cable lattice in Figure 11.

4.2 Fair Isle Colorwork

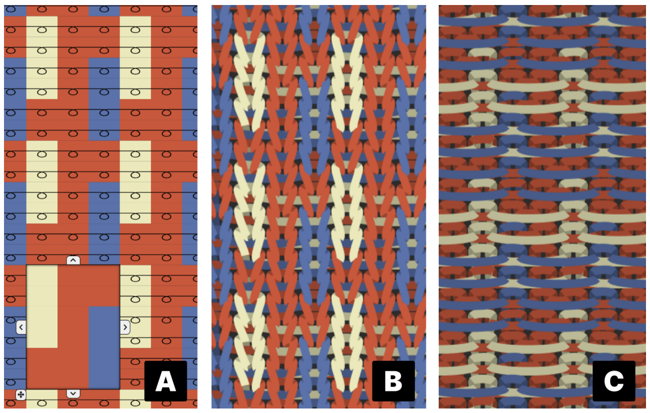

Figure 14: Colorwork: We support colorwork patterns which include more than two colors per row, such as this motif which includes three colors per row (A). The fabric is shown in the yarn visualization (B), and the resulting floats can be inspected by viewing the back (C).

Fair Isle colorwork (also known as stranded colorwork or float jacquard) is a technique for producing color patterns where multiple yarns are used across a whole row of knitting. For each row, the yarn which should appear on the front of the fabric knits, and the non-visible yarn (or yarns) floats behind the knit stitches. Figure 12 shows an example where multiple colorwork motifs are used to pattern a "doodle cowl", a type of pattern which is currently popular in the hand knitting community as a sampler of themed colorwork motifs. Our yarn separation can theoretically handle an arbitrary number of yarns per row, although there are practical limitations to this depending on maximum float length. Figure 14 shows a colorwork pattern which includes three colors per row. The back of the fabric can be examined in the yarn simulation, which enables inspection of the resulting floats.

4.3 Intarsia Colorwork

Intarsia is a colorwork technique appropriate for patterns which include large, unbroken blocks of color. A separate yarn feed is used to knit each segment of color which appears in a row. This means that a circle which appears in the center of a fabric requires three separate yarns (Figure 8), and the "U" in UIST requires five separate yarns (Figure 15). Our editor supports design of intarsia patterns via yarn paths and yarn fills. To prevent gaps from forming between adjacent blocks of color, the yarns must be joined at the color changes. One strategy for joining is to add tucks to the preceding color segment. In our editor, join conditions are specified

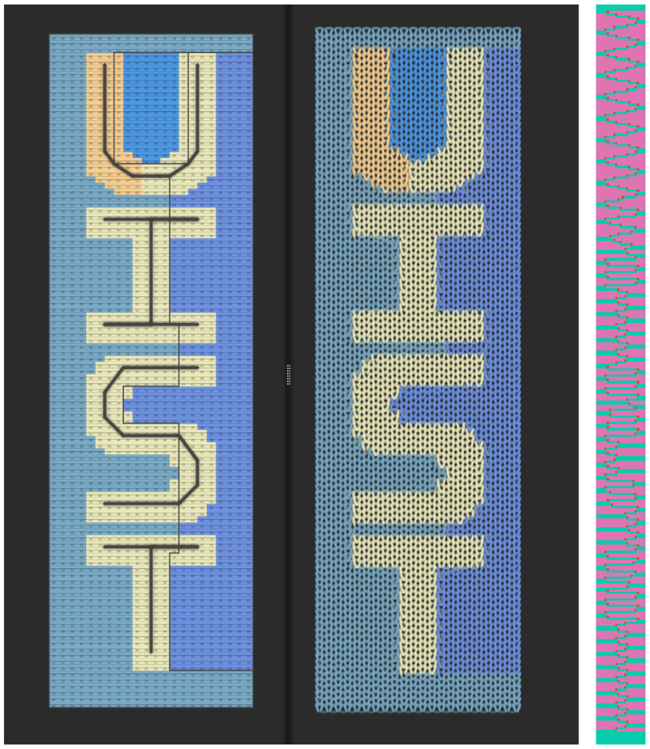

Figure 15: Path Intarsia: Whereas Figure 8 shows a patch defined with a circle boundary, this example uses a path with a corresponding stitch block to create the letters UIST. "U" in "UIST" requires five separate yarns to knit it with the intarsia technique: three in the background color (one for each side plus one in the middle) and two for the letter color (one for each leg).

by selecting the "tucks" join mode option on a boundary or path (Figure 9). By placing a tuck on either the front or back bed depending on the position of the joined stitch, we also support design of textured intarsia patterns.

4.4 Lace

Lace is knit by using different operations such as transfers to create delicate patterns of holes. Blended primitives offer multiple approaches to the design of lace patterns, which can be specified using a traditional block-based repeat as a stitch fill of a boundary-defined region (Figure 13, left) or by using a stitch path to tile a block of instructions along a vector-based path (Figure 13, right). The stitch fill approach may be more appropriate for all-over textures, while stitch paths are better suited to the design of individual motifs.

5 FURTHER APPLICATIONS: SENSORS AND FOLDS

Blended primitives enable us to build abstractions for applications beyond the more traditional knitting techniques described in the

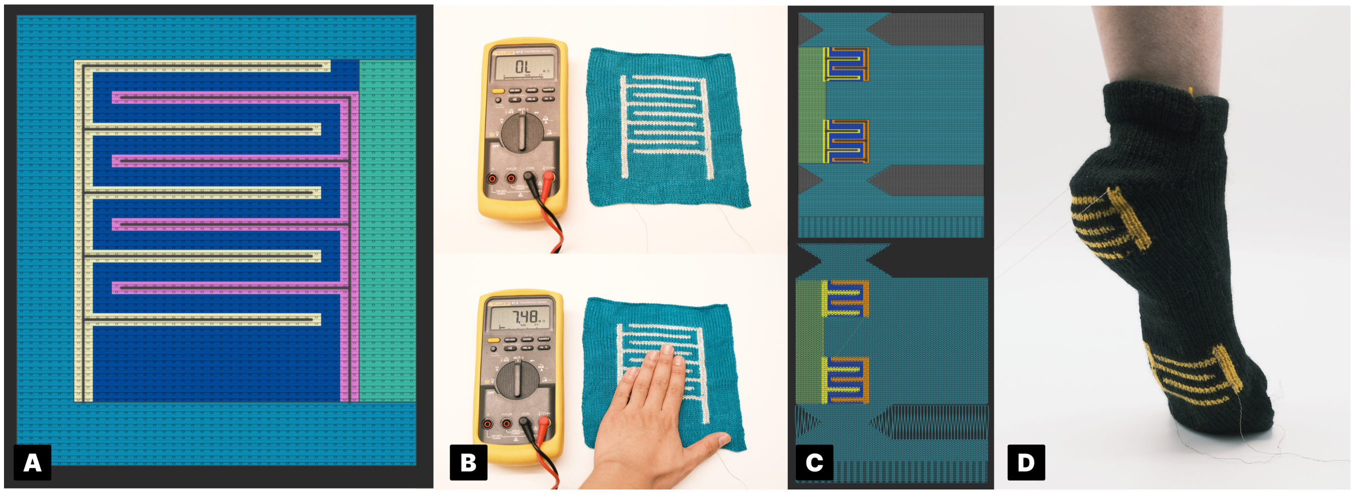

Figure 16: Knit pressure sensors: Yarn paths and fills can be used to design an interdigital electrode structure in our editor, which can be knit via the intarsia technique (A). Note that we have used contrasting colors for the five separate yarns for visualization purposes, but all yarns could be the same color when knit to hide the presence of the sensor. The interdigital electrode structure can be used to sense pressure when in direct contact with skin (B), and positioned on a sock pattern in our editing environment (C, top) and visualized (C, bottom). Stretched loops in the yarn visualization indicate loop connectivity across short rows required to shape the sock heel. The knit sock (D) can sense changes in pressure at the heel and toe.

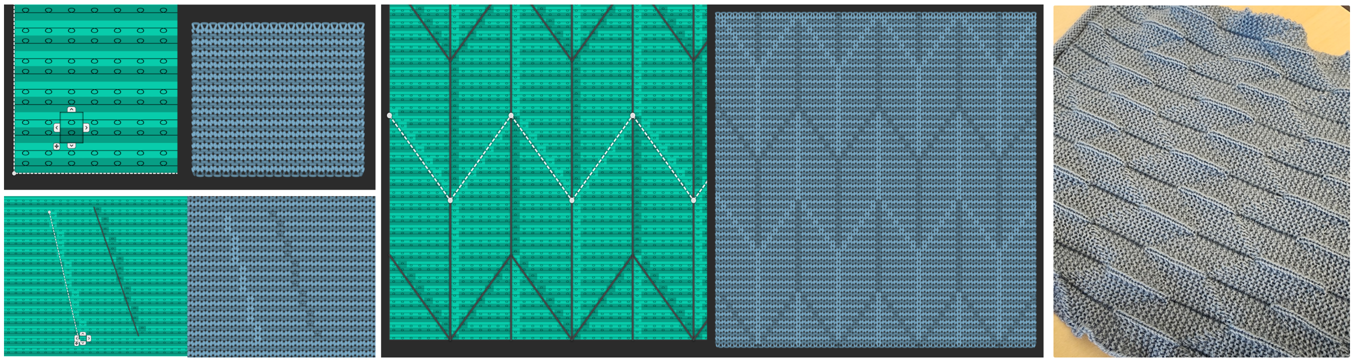

Figure 17: Self-folding knit pleats: Origami pleat patterns can be defined by layering knit and purl stitch paths over a garter stitch region. This takes advantage of different stitchs' tendency to deform to create self-folding patterns.

previous section. We describe two example applications in this section: intarsia-based sensing structures and origami-inspired pleat patterns.

5.1 Origami-Inspired Pleat Patterns

One technique to design knitted folds or pleats is to take advantage of the curling behavior inherent to knit fabric: surfaces made entirely of knit stitches have a natural tendency to curl towards the back. Staggering knit and purl stitches can even out this curling behavior and make a fabric lie flat. Therefore, to design a pleat, we can use a base pattern with minimal curl and strategically place knit or purl stitches where we want the fabric to curl. Blended primitives enables us to abstract this core idea into reusable abstractions for mountain and valley folds, which we have used to design origami pleat patterns (Figure 17). Our origami patterns consist of a background boundary with a garter stitch fill (alternating rows of knit and purl stitches). We then define a mountain fold as a knit stitch path, and a valley fold as a purl stitch path. Using these stitch paths, we can draw mountain and valley folds atop the garter background. This shows how our tool supports the design of folding and shape-changing knits.

5.2 Intarsia-based Sensing Structures

Knit textiles can be used to sense environmental changes by incorporating conductive and other active yarns into the fabric structure. These sensing structures can be used in a variety of ways to sense stretch, pressure, touch, humidity, and more. This has been an area of increasing interest to researchers exploring the intersection of HCI and wearable interfaces [3, 42, 46, 51, 54, 65]. However, the challenges of low-level knit programming extends to this emerging

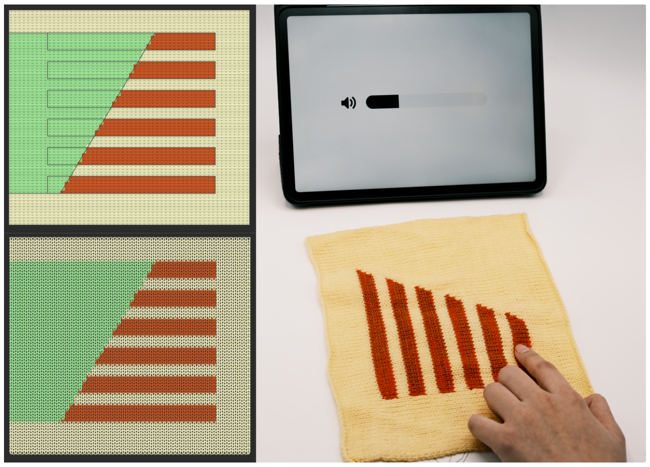

Figure 18: Knit sensors: Knitting an intarsia pattern with conductive yarns creates distinct conductive regions that can be used for capacitive touch sensing.

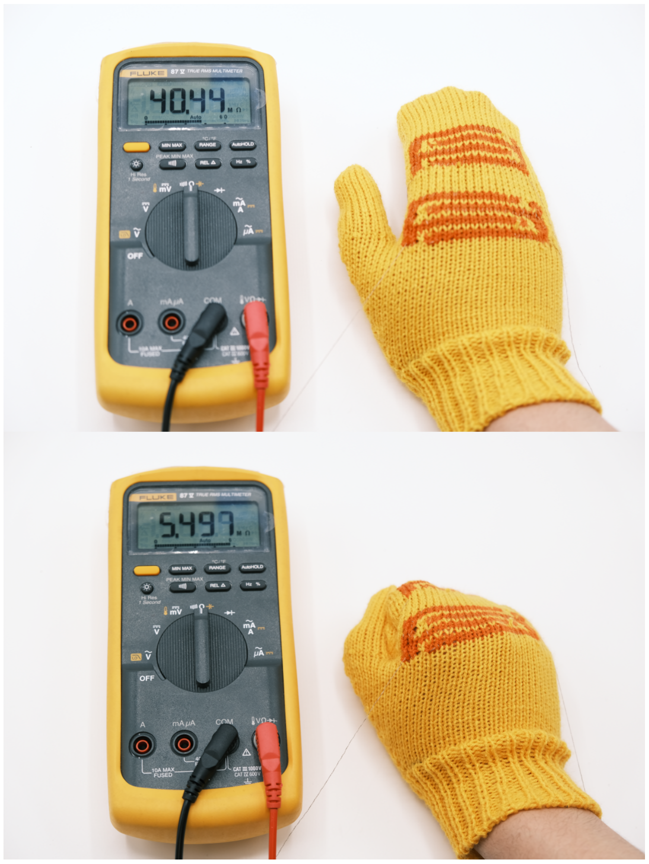

Figure 19: Knit bend sensors: The interdigital electrode structure used for the sock shown in Figure 16 can also be used to sense stretch, used here as part of a bend-sensing mitten.

domain. Our design tool is not specific to the design of knit sensors, but our yarn paths and fills support the design of single-layer multi-yarn sensing structures based on the intarsia technique.

We demonstrate how to knit two types of intarsia-based sensors: capacitive touch sensors, which we use to create the volume control pad in Figure 18, and interdigital electrode sensors, which we use to sense pressure (to detect foot posture with a pressure-sensing sock in Figure 16) and stretch/bend (used in the bend-sensing mitten in Figure 19). For the latter sensor type, interdigital electrodes are positioned against the skin, where higher pressure, resulting either from direct contact (sock) or from localized bending (mitten), will augment skin-electrode contact and thus reduce the resistance measured across the electrodes. We knit these samples by running a conductive embroidery thread (Madeira HC-40) in the same yarn feed as the yarns which were intended to be conductive. This shows how our design tool supports the integration of yarns into localized sensor regions.

6 SYSTEM IMPLEMENTATION

The blended primitives design tool is browser-based and written in client-side JavaScript. For a link to the open-source code repository and hosted version of the tool, please visit the project page: https://depts.washington.edu/machines/projects/blended-primitives/. The tool consists of a split-pane view of our novel pattern-editing interface and a 2D yarn-level simulation based on the open-source KnitScape system [64]. In this section we describe our implementation of a novel pattern editing interface for design with blended raster/vector primitives as well as our extensions to the KnitScape yarn model and simulation. We tested our export format on the Kniterate machine, which supports import of an instruction chart and associated yarn sequence.

6.1 Stitch and Yarn Chart Rasterization

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

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

To generate the machine instructions and yarn topology for the yarn simulation, each yarn must be scheduled in its own carriage pass. This requires further processing of the stitch and yarn charts and is handled during our yarn separation and transfer planning step described in Section 6.2.

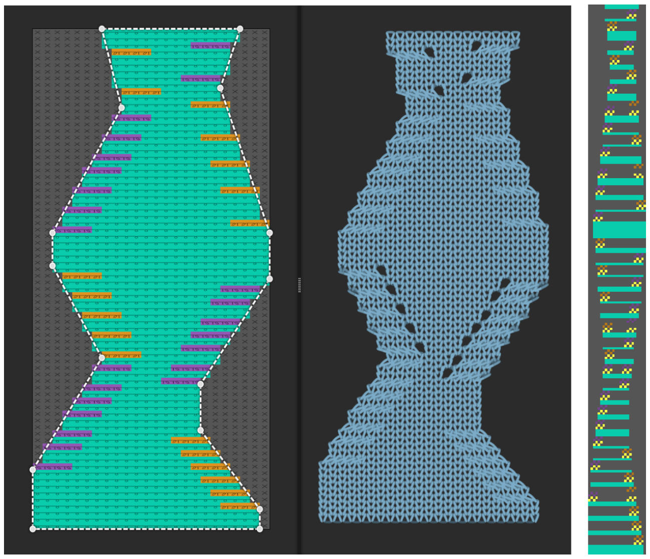

Figure 20: When a boundary includes shaping, fashioning instructions are written where rows increase and decrease and update automatically as the boundary is reshaped. Pictured here is an oddly-shaped boundary in the charting interface (left), the resulting yarn simulation (center), and the time-needle view which includes scheduled transfers (right).

6.1.1 Boundary Rasterization. We rasterize each polyline boundary using a scanline fill approach modified to incorporate some knitting-specific considerations. Pseudocode can be found in Appendix C. At a high level we follow a standard scanline polygon fill algorithm [21], initializing an edge table (ET) and active edge table (AET) and processing the intersections between a horizontal scanline and any active edges in pairs. In order to better approximate the boundary, we use a scanline located halfway up the cell ($y + 0.5$). We do this by offsetting the edge's current x value during edge table creation, setting it to $x1 + (dx/2)$. For boundaries that have shaping instructions, each edge also stores the last x value rounded to the nearest integer ($xLast$), which we examine to detect when rows are increasing or decreasing.

During the main loop, we round the scanline intersection of each edge pair to the nearest integer to find the start and end points for the pattern fill ($x1$ and $x2$). We then fill the space between $x1$ and $x2$ in the stitch and yarn charts by copying values from the respective blocks, skipping locations where the block includes the "transparent" stitch or yarn. The interior space is patterned using a simple 2d repeat of the base block. It would be straightforward to extend support to other repeat modes (e.g., ones that include block reflection or an offset between pattern rows or columns). If the boundary has shaping instructions, we examine the relationship between $x1$ and $x2$ and the $lastX$ value of their respective edges and write fashioning instructions (transfers) to the prior row to account for a difference: either moving a group of loops out (an increase) or in (a decrease). Shaping instructions are only applied to single-stitch increases and decreases, i.e., edges where $-1 \leq 1/m \leq 1$.

6.1.2 Path Rasterization. We rasterize each polyline path using a Bresenham approach [7], again modified to incorporate some knitting-specific considerations. Pseudocode can be found in Appendix C. We traverse each polyline segment from $[x0, y0]$ to $[x1, y1]$ and draw the associated stitch and yarn blocks depending on the tile mode of the path.

The "tiled" mode examines the last position of the block and only plots it if the difference in x or y is greater than the block width or height, respectively (an example of this can be seen in Figure 10-B). The "dx" and "dy" modes specify whether the tile spacing should iterate in the horizontal or vertical direction. This can be helpful in cases where block iterations require exact spacing over a number of rows (or columns), but the other dimension does not matter. Finally, the "overlap" mode plots the whole block at every location.

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

6.2 Yarn Separation and Transfer Planning

When machine knitting complex patterns, loops often must be transferred from a front bed to a back bed and vice versa. We handle transfer planning similarly to the schoolbus approach for the flat lace transfer problem described by Lin et al. [40]. Full implementation can be found in our source code repository. However, instead of transferring all loops between beds, we only transfer those with target offsets. A simple example demonstrating our transfer planner can be seen in Figure 21. First, we transfer all loops with a target offset to the back bed (bright yellow), staggering them so as to not transfer adjacent loops in the same pass of the carriage. Then, for each target offset (in this case -1 and 1) we transfer the loops back to the main bed, again staggering them so as to not transfer adjacent loops.

6.3 Yarn Topology and Simulation

Our yarn simulation extends the open-source KnitScape system [64], which was developed for the design and simulation of slip and tuck colorwork patterns. The yarn simulation uses an approach proposed by Kapllani et al. [32] to generate a graph of yarn topology from a chart of knitting instructions that encodes the order in which a yarn visits contact nodes where it is intertwined. To draw the yarns, leg and loop segments of a spline which is traced around the contact nodes are separated onto layers and shaded to give the illusion of depth. To preview the deformation caused by loops which have been stretched across multiple rows of knitting, the visualization also includes a particle-spring simulation where each yarn segment tries to contract to a rest length based on the distance it extends from its original row of instructions. We made a number of extensions to the topology model and yarn simulation in order

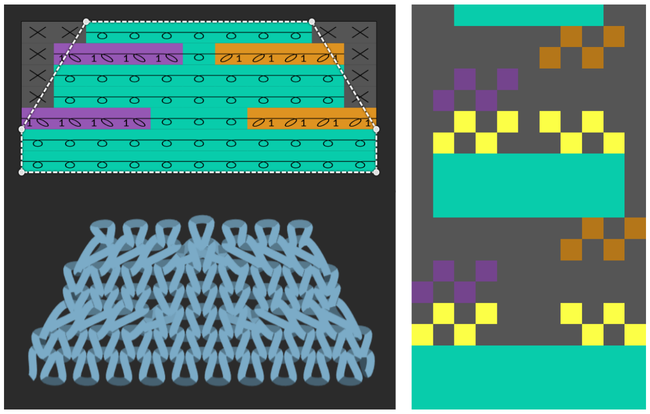

Figure 21: We stagger loop transfers so as to not transfer adjacent loops in the same pass of the carriage.

to support patterns which include transfers (e.g., cables, lace, and fully-fashioned shaping), and patterns which include multiple yarns per row (e.g., intarsia or fair isle colorwork).

6.3.1 Composite Transfers. Extending the existing knit, purl, slip, and tuck operation set, we implemented a composite "knit then transfer" operation where a loop is knit on the front or back bed and will be moved to its target location on the next transfer pass. Our full operation set can be seen in Figure 3. Currently, we have not implemented commands which enable control over the order in which loops end up in their target location (which can affect surface appearance in some cases), and this is determined by carriage pass direction. We think it would be straightforward to implement additional composite operations which enable finer control over transfer planning, and that this should be a priority for future work. In order to visualize the greater depth introduced by a transferred loop, we also extended the existing yarn visualization to support an arbitrary number of shaded layers determined by the maximum stack of yarns at any location in a pattern. To determine the correct stacking order for transferred loops, we implemented the approach described by Kapllani et al. [31]. Our full implementation is included in our open-source code repository.

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

6.4 Challenges and Opportunities

Our demonstration examples show that a 2D chart-based notation such as ours can be powerful for designing mid-level texture and colorwork effects, but some design tasks (like 3D shaping) are challenging due to the convoluted relationship between a 2D notation and 3D result. While it is possible to use our notation to design surfaces with short row shaping (such as the sock in Figure 16), a 3D shaping primitive like a tube [45] or mesh [50] may be better suited to that task. However, changes to fabric structure can significantly impact the fabric gauge (e.g., some textures are short and wide while others are tall and narrow), which may need to be accounted for in the overall shape specification. This poses a practical challenge when building knit design interfaces: overall shape, colorwork/texture patterning, *and* the interaction between them determine the aesthetic and functional qualities of the final knit.

Future design tools must investigate how to support creative exploration of these different dimensions–and their interactions–in tandem. We don't think that future knitting design tools should prioritize reconciling disparate, task-specific representations under one notation system, as we think such a system would risk supporting some tasks far better than others (or worse: being "bad at everything"). We think future interfaces should instead focus on enabling someone to visualize and rapidly work *across* task-specific notations, such as by providing multiple synchronized editing views. As an example, we can imagine a future version of our system which includes a 3D stitch mesh editing pane, in which a stitch mesh [50] could be used to drive a base "shaping boundary" in our chart editor. Stitch and yarn paths could then be defined in the chart editor and visualized on the mesh surface to give a sense of their position in the global fabric. Another way to accomplish this in the context of our current interface would be to develop domain-specific block editors which can compile to our charting language. For example, a "brioche editor" which uses Albaugh et al. [2]'s brioche notation could be opened in place of our current stitch block editor.

While we have not formally tested the limits of our approach for fabric size, our tool is performant (i.e., changes to the chart update the visualization with minimal delay [<50ms]) on our setup (Chromium on an i7 laptop with integrated graphics) for charts below 200x200 cells. Currently, update lag is largely due to a bottleneck when performing local searches to find the origin of unanchored yarn contacts during yarn topology generation, which particularly impacts intarsia patterns (a future iteration of our system may improve this by storing loop origin). However, we have tried to mitigate the impact of this delay on user experience by offloading computation to web workers where possible and adding a setting to toggle whether the visualization/yarn topology is updated on-the-fly (on input to the chart). We intend to continue to improve our simulation by incorporating advances made in recent work on efficient yarn-level simulation.

7 DISCUSSION

Machine knitting is an incredibly powerful domain with a massive design space. In order to explore this design space, we need appropriate design representations that enable us to quickly iterate on various aspects of the knit object of interest. Our work focuses on supporting the design of new knit surface patterning effects such as cable structures, colorwork, and texture patterns. This is how our work differs from prior contributions which have largely focused on capturing overall fabric shape, such as tubes and sheets [34, 35, 45] and stitch meshes [49, 50]. We argue that design of fabric-level effects such as cables is a fundamentally different task than design of three-dimensional shaping, which has been the focus of prior work. We think that because such fabric structures deal closely with material behavior on the magnitude of an individual stitch, they require design representations which maintain easy access to low-level stitch operations while also supporting rapid manipulation of higher-level operation groups. We have implemented a design representation for knitting patterns we term *blended primitives*, which we contribute alongside our design tool.

Future directions for our work fall into three high-level categories, each of which is relevant to research in and beyond HCI. First, expanding the space of knit objects that can be specified and visualized in our system will be important to enable design of complex fabric structures, supporting ongoing development of novel functional fabrics. Currently, only patterned sheets can be designed in our editor. Through examples, we have demonstrated that a variety of single-layer sensing structures can be designed this way via intarsia patterning. However, due to a limitation of our underlying topology model we do not support double-sided or circular-knit fabrics, techniques which have been previously demonstrated to also be effective methods for fabricating knit sensors and actuators [3, 36, 42, 43, 54, 55]. We think it will be feasible to extend our editor to these areas with reasonable effort, and this is a priority for future work.

Second, providing designers with many different ways to specify and manipulate primitive elements will be crucial for those exploring the intersection of machine knitting and other domains. We were interested in building a (somewhat) vector-based design environment for knitting in part due to the many possible avenues for interoperability with existing tools and file formats. For example, we think it would be straightforward to implement support for a crease pattern format (such as [12]) that automatically defines stitch paths along fold creases, enabling knit origami patterns (such as the one in Figure 17) to be specified in origami-specific design tools and imported into our system. We anticipate that keeping an eye towards interoperability in future tool development will be helpful to those venturing into domains with few established design tools, such as shape-changing fibers [14].

Third, informing a designer of fabrication constraints and errors during design will always be an important aspect of a design tool. We think that the yarn-level visualization included in our tool is effective at helping designers navigate some of the challenges of low-level knit programming, such removing long floats in the example intarsia workflow shown in Figure 8. We think that the goals of a simulation used alongside a charting environment are different from those focused solely on rendering, and should be centered on providing effective feedback to the designers. For example, rendering fiber-level detail for more "realistic" yarns might distract from features of yarn topology which may be more relevant to designers, such as loop connectivity and float position.

Finally, while we have focused on design for machine knitting in this paper, we want to recognize its deep roots in the craft in hand knitting, which continues to be beloved by millions around the globe. Hand knitting-oriented design and simulation tools would have the potential to reach and delight a massive audience. Our charting language was designed with machine knitting in mind, but it already is compatible with a significant subset of hand-knit charting notation, which has had hundreds of years to mature and continues to be the primary form of visual pattern sharing among hand knitters [8]. We are very interested in building a dedicated hand-knitting version of our system, as we think that this would not only be a good way for hand knitters to take advantage of our blended primitives editing tools and synchronized yarn visualization, but it could also provide an alternative way to edit machine knitting patterns that would be accessible to those who are already fluent in chart notation.

Some related work has already begun in this space, e.g. by formalizing knitting shorthand [22]. The possibility space of hand knitting is far greater than that of machine knitting, so development of design tools that include some form of simulation presents some practical development hurdles and open research problems, although some mesh-based approaches exist [25]. However, such tools would also provide us with an easy way to import, visualize, and archive historic charted patterns. This could provide an important cultural snapshot of a craft which, despite its undeniable ubiquity, is often overlooked [44, 63].

8 CONCLUSION

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

ACKNOWLEDGMENTS

Special thanks to Afroditi Psarra and the UW DXARTS Softlab for letting us use their Kniterate machine. This research is partly supported by the NSF through awards 2339273 and 2047342.

REFERENCES

[1] Roland Aigner, Mira Alida Haberfellner, and Michael Haller. 2024. Loopsense: Low-Scale, Unobtrusive, and Minimally Invasive Knitted Force Sensors for Multi-Modal Input, Enabled by Selective Loop-Meshing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3613904.3642528

[2] Lea Albaugh, Scott E Hudson, and Lining Yao. 2023. Physically Situated Tools for Exploring a Grain Space in Computational Machine Knitting. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. ACM, Hamburg Germany, 1–14. https://doi.org/10.1145/3544548.3581434
[3] Lea Albaugh, James McCann, Scott E. Hudson, and Lining Yao. 2021. Engineering Multifunctional Spacer Fabrics Through Machine Knitting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–12.
[4] AYAB Developers. 2024. AYAB - All Yarns Are Beautiful. https://ayab-knitting.com/, accessed 2024.
[5] Emmanuel Ayodele, Syed Ali Raza Zaidi, Jane Scott, Zhiqiang Zhang, Maryam Hafeez, and Des McLernon. 2021. The Effect of Miss and Tuck Stitches on a Weft Knit Strain Sensor. *Sensors* 21, 2 (Jan. 2021), 358. https://doi.org/10.3390/s21020358
[6] David E. Breen, Donald H. House, and Michael J. Wozny. 1994. Predicting the Drape of Woven Cloth Using Interacting Particles. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '94)*. Association for Computing Machinery, New York, NY, USA, 365–372. https://doi.org/10.1145/192161.192259
[7] J. E. Bresenham. 1965. Algorithm for computer control of a digital plotter. *IBM Systems Journal* 4, 1 (1965), 25–30. https://doi.org/10.1147/sj.41.0025
[8] J. C. Briar. 2011. *Charts Made Simple: Understanding Knitting Charts Visually : A Knitting on Paper Book.* Glass Iris Publications.
[9] Juan J. Casafranca, Gabriel Cirio, Alejandro Rodríguez, Eder Miguel, and Miguel A. Otaduy. 2020. Mixing Yarns and Triangles in Cloth Simulation. *Computer Graphics Forum* (2020). https://doi.org/10.1111/cgf.13915
[10] Carlos Castillo, Jorge López-Moreno, and Carlos Aliaga. 2019. Recent advances in fabric appearance reproduction. *Computers & Graphics* 84 (2019), 103–121. https://doi.org/10.1016/j.cag.2019.07.007
[11] Gabriel Cirio, Jorge Lopez-Moreno, and Miguel A. Otaduy. 2017. Yarn-Level Cloth Simulation with Sliding Persistent Contacts. *IEEE Transactions on Visualization and Computer Graphics* 23, 2 (2017), 1152–1162. https://doi.org/10.1109/TVCG.2016.2592908
[12] Erik D Demaine, Jason S Ku, and Robert J Lang. [n. d.]. A New File Standard to Represent Folded Structures. ([n. d.]).
[13] Laura Devendorf, Kathryn Walters, Marianne Fairbanks, Etta Sandry, and Emma R Goodwill. 2023. AdaCAD: Parametric Design as a New Form of Notation for Complex Weaving. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems (CHI '23)*. Association for Computing Machinery, New York, NY, USA, 1–18. https://doi.org/10.1145/3544548.3581571
[14] Jack Forman, Ozgun Kilic Afsar, Sarah Nicita, Rosalie Hsin-Ju Lin, Liu Yang, Megan Hofmann, Akshay Kothakonda, Zachary Gordon, Cedric Honnet, Kristen Dorsey, Neil Gershenfeld, and Hiroshi Ishii. 2023. FibeRobo: Fabricating 4D Fiber Interfaces by Continuous Drawing of Temperature Tunable Liquid Crystal Elastomers. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. Association for Computing Machinery, New York, NY, USA, 1–17. https://doi.org/10.1145/3586183.3606732
[15] Mikhaila Friske, Shanel Wu, and Laura Devendorf. 2019. AdaCAD: Crafting Software For Smart Textiles Design. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems (CHI '19)*. Association for Computing Machinery, New York, NY, USA, 1–13.
[16] Timothy Gerstner, Doug DeCarlo, Marc Alexa, Adam Finkelstein, Yotam Gingold, and Andrew Nealen. 2012. Pixelated Image Abstraction. In *International Symposium on Non-Photorealistic Animation and Rendering*, Paul Asente and Cindy Grimm (Eds.). The Eurographics Association. https://doi.org/10.2312/PE/NPAR/NPAR12/029-036
[17] Maas Goudswaard, Abel Abraham, Bruna Goveia da Rocha, Kristina Andersen, and Rong-Hao Liang. 2020. *FabriClick: Interweaving Pushbuttons into Fabrics Using 3D Printing and Digital Embroidery.* 393 pages. https://doi.org/10.1145/3357236.3395569
[18] Bruna Goveia Da Rocha, Oscar Tomico, Panos Markopoulos, and Daniel Tetteroo. 2020. Crafting Research Products through Digital Machine Embroidery. In *Proceedings of the 2020 ACM Designing Interactive Systems Conference*. ACM, Eindhoven Netherlands, 341–350. https://doi.org/10.1145/3357236.3395443
[19] Susan Guagliumi. 1990. *Hand-Manipulated Stitches for Machine Knitters.* GUAGLIUMIDOTCOM.
[20] Nur Al-huda Hamdan, Simon Voelker, and Jan Borchers. 2018. Sketch&Stitch: Interactive Embroidery for E-textiles. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems (CHI '18)*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3173656
[21] Donald Hearn, Pauline Baker, and Warren Carithers. 2010. *Computer Graphics with Open GL* (fourth ed.). Pearson.
[22] Megan Hofmann, Lea Albaugh, Ticha Sethapakadi, Jessica Hodgins, Scott E. Hudson, James McCann, and Jennifer Mankoff. 2019. KnitPicking Textures: Programming and Modifying Complex Knitted Textures for Machine and Hand Knitting. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 5–16. https://doi.org/10.1145/3332165.3347886
[23] Megan Hofmann, Lea Albaugh, Tongyan Wang, Jennifer Mankoff, and Scott E Hudson. 2023. KnitScript: A Domain-Specific Scripting Language for Advanced Machine Knitting. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology (UIST '23)*. Association for Computing Machinery, New York, NY, USA, 1–21. https://doi.org/10.1145/3586183.3606789
[24] Megan Hofmann, Jennifer Mankoff, and Scott E. Hudson. 2020. KnitGIST: A Programming Synthesis Toolkit for Generating Functional Machine-Knitting Textures. In *Proceedings of the 33rd Annual ACM Symposium on User Interface Software and Technology (UIST '20)*. Association for Computing Machinery, New York, NY, USA, 1234–1247. https://doi.org/10.1145/3379337.3415590
[25] Yuki Igarashi, Takeo Igarashi, and Hiromasa Suzuki. 2008. Knitty: 3D Modeling of Knitted Animals with a Production Assistant Interface. (2008). https://doi.org/10.2312/egs.20081011
[26] Tiffany C. Inglis and Craig S. Kaplan. 2012. Pixelating vector line art. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (Annecy, France) *(NPAR '12)*. Eurographics Association, Goslar, DEU, 21–28.
[27] Tiffany C. Inglis, Daniel Vogel, and Craig S. Kaplan. 2013. Rasterizing and antialiasing vector line art in the pixel art style. In *Proceedings of the Symposium on Non-Photorealistic Animation and Rendering* (Anaheim, California) *(NPAR '13)*. Association for Computing Machinery, New York, NY, USA, 25–32. https://doi.org/10.1145/2486042.2486044
[28] Benjamin Jones, Yuxuan Mei, Haisen Zhao, Taylor Gotfrid, Jennifer Mankoff, and Adriana Schulz. 2022. Computational Design of Knit Templates. *ACM Transactions on Graphics* 41, 2 (April 2022), 1–16. https://doi.org/10.1145/3488006
[29] Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2008. Simulating Knitted Cloth at the Yarn Level. In *ACM SIGGRAPH 2008 Papers* (Los Angeles, California) *(SIGGRAPH '08)*. Association for Computing Machinery, New York, NY, USA, Article 65, 9 pages. https://doi.org/10.1145/1399504.1360664
[30] Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2010. Efficient Yarn-Based Cloth with Adaptive Contact Linearization. *ACM Trans. Graph.* 29, 4, Article 105 (jul 2010), 10 pages. https://doi.org/10.1145/1778765.1778842
[31] Levi Kapllani, Chelsea Amanatides, Genevieve Dion, and David E. Breen. 2022. Loop Order Analysis of Weft-Knitted Textiles. *Textiles* 2, 2 (June 2022), 275–295. https://doi.org/10.3390/textiles2020015
[32] Levi Kapllani, Chelsea Amanatides, Genevieve Dion, Vadim Shapiro, and David E. Breen. 2021. TopoKnit: A Process-Oriented Representation for Modeling the Topology of Yarns in Weft-Knitted Textiles. *Graphical Models* 118 (Nov. 2021), 101114. https://doi.org/10.1016/j.gmod.2021.101114
[33] Ayelet Karmon, Yoav Sterman, Tom Shaked, Eyal Sheffer, and Shoval Nir. 2018. KNITIT: A Computational Tool for Design, Simulation, and Fabrication of Multiple Structured Knits. In *Proceedings of the 2nd ACM Symposium on Computational Fabrication*. ACM, Cambridge Massachusetts, 1–10. https://doi.org/10.1145/3213512.3213516
[34] Alexandre Kaspar, Liane Makatura, and Wojciech Matusik. 2019. Knitting Skeletons: A Computer-Aided Design Tool for Shaping and Patterning of Knitted Garments. In *Proceedings of the 32nd Annual ACM Symposium on User Interface Software and Technology (UIST '19)*. Association for Computing Machinery, New York, NY, USA, 53–65. https://doi.org/10.1145/3332165.3347879
[35] Alexandre Kaspar, Kui Wu, Yiyue Luo, Liane Makatura, and Wojciech Matusik. 2021. Knit Sketching: From Cut & Sew Patterns to Machine-Knit Garments. *ACM Transactions on Graphics* 40, 4 (Aug. 2021), 1–15. https://doi.org/10.1145/3450626.3459752
[36] Jin Hee (Heather) Kim, Kunpeng Huang, Simone White, Melissa Conroy, and Cindy Hsin-Liu Kao. 2021. KnitDermis: Fabricating Tactile On-Body Interfaces Through Machine Knitting. In *Proceedings of the 2021 ACM Designing Interactive Systems Conference (DIS '21)*. Association for Computing Machinery, New York, NY, USA, 1183–1200. https://doi.org/10.1145/3461778.3462007
[37] Bill King. 2023. *Machine Knitting Techniques: Cables.* Crowood Press.
[38] Jonathan Leaf, Rundong Wu, Eston Schweickart, Doug L. James, and Steve Marschner. 2018. Interactive Design of Periodic Yarn-Level Cloth Patterns. *ACM Trans. Graph.* 37, 6, Article 202 (dec 2018), 15 pages. https://doi.org/10.1145/3272127.3275105
[39] Jenny Lin, Vidya Narayanan, Yuka Ikarashi, Jonathan Ragan-Kelley, Gilbert Bernstein, and James McCann. 2023. Semantics and Scheduling for Machine Knitting Compilers. *ACM Transactions on Graphics* 42, 4 (Aug. 2023), 1–26. https://doi.org/10.1145/3592449
[40] Jenny Lin, Vidya Narayanan, and James McCann. 2018. Efficient Transfer Planning for Flat Knitting. In *Proceedings of the 2nd ACM Symposium on Computational Fabrication (SCF '18)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/3213512.3213515
[41] Zishun Liu, Xingjian Han, Yuchen Zhang, Xiangjia Chen, Yu-Kun Lai, Eugeni L. Doubrovski, Emily Whiting, and Charlie C. L. Wang. 2021. Knitting 4D Garments with Elasticity Controlled for Body Motion. *ACM Transactions on Graphics* 40, 4 (July 2021), 62:1–62:16. https://doi.org/10.1145/3450626.3459868
[42] Yiyue Luo, Kui Wu, Tomás Palacios, and Wojciech Matusik. 2021. KnitUI: Fabricating Interactive and Sensing Textiles with Machine Knitting. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Number 668 in CHI '21. Association for Computing Machinery, New York, NY, USA, 1–12.

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

[44] Anne L. MacDonald. 2010. *No Idle Hands: The Social History of American Knitting*. Random House Publishing Group.

[45] James McCann, Lea Albaugh, Vidya Narayanan, April Grow, Wojciech Matusik, Jennifer Mankoff, and Jessica Hodgins. 2016. A Compiler for 3D Machine Knitting. *ACM Transactions on Graphics* 35, 4 (July 2016), 49:1–49:11. https://doi.org/10.1145/2897824.2925940

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

[47] M. Meißner and B. Eberhardt. 1998. The Art of Knitted Fabrics, Realistic & Physically Based Modelling of Knitted Patterns. *Computer Graphics Forum* 17, 3 (1998), 355–362. https://doi.org/10.1111/1467-8659.00282 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/1467-8659.00282

[48] Rahul Mitra, Liane Makatura, Emily Whiting, and Edward Chien. 2023. Helix-Free Stripes for Knit Graph Design. In *ACM SIGGRAPH 2023 Conference Proceedings* (, Los Angeles, CA, USA,) *(SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 75, 9 pages. https://doi.org/10.1145/3588432.3591564

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

[50] Vidya Narayanan, Kui Wu, Cem Yuksel, and James McCann. 2019. Visual Knitting Machine Programming. *ACM Transactions on Graphics* 38, 4 (July 2019), 63:1–63:13. https://doi.org/10.1145/3306346.3322995

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

[52] M. L. V. Pitteway. 1967. Algorithm for drawing ellipses or hyperbolae with a digital plotter. *Comput. J.* 10, 3 (01 1967), 282–289. https://doi.org/10.1093/comjnl/10.3.282 arXiv:https://academic.oup.com/comjnl/article-pdf/10/3/282/1333509/100282.pdf

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

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

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

[56] Davi Post. 2024. Img2track. https://daviworks.com/knitting/, accessed 2024.

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

[58] Rosa M. Sánchez-Banderas, Alejandro Rodríguez, Héctor Barreiro, and Miguel A. Otaduy. 2020. Robust Eulerian-on-Lagrangian Rods. *ACM Trans. Graph.* 39, 4 (Aug. 2020), 59:59:1–59:59:10. https://doi.org/10.1145/3386569.3392489

[59] Shneiderman. 1983. Direct Manipulation: A Step Beyond Programming Languages. *Computer* 16, 8 (Aug. 1983), 57–69. https://doi.org/10.1109/MC.1983.1654471

[60] David J. Spencer. 2001. *Knitting Technology: A Comprehensive Handbook and Practical Guide*. CRC Press.

[61] Georg Sperl, Rahul Narain, and Chris Wojtan. 2021. Mechanics-Aware Deformation of Yarn Pattern Geometry. *ACM Trans. Graph.* 40, 4 (July 2021), 168:1–168:11. https://doi.org/10.1145/3450626.3459816

[62] Georg Sperl, Rosa M. Sánchez-Banderas, Manwen Li, Chris Wojtan, and Miguel A. Otaduy. 2022. Estimation of Yarn-Level Simulation Models for Production Fabrics. *ACM Trans. Graph.* 41, 4 (July 2022), 65:1–65:15. https://doi.org/10.1145/3528223.3530167

[63] Joanne Turney. 2009. *The Culture of Knitting*. Berg Publishers.

[64] Hannah Twigg-Smith, Emily Whiting, and Nadya Peek. 2024. KnitScape: Computational Design and Yarn-Level Simulation of Slip and Tuck Colorwork Knitting Patterns. In *Proceedings of the CHI Conference on Human Factors in Computing Systems (CHI '24)*. Association for Computing Machinery, New York, NY, USA, 1–20. https://doi.org/10.1145/3613904.3642799

[65] Richard Vallett, Denisa Qori McDonald, Genevieve Dion, Youngmoo Kim, and Ali Shokoufandeh. 2020. Toward Accurate Sensing with Knitted Fabric: Applications and Technical Considerations. *Proceedings of the ACM on Human-Computer Interaction* 4, EICS (June 2020), 79:1–79:26. https://doi.org/10.1145/3394981

[66] Jerry Van Aken and Mark Novak. 1985. Curve-drawing algorithms for Raster displays. *ACM Trans. Graph.* 4, 2 (apr 1985), 147–169. https://doi.org/10.1145/282918.282943

[67] Kui Wu, Hannah Swan, and Cem Yuksel. 2019. Knittable Stitch Meshes. *ACM Transactions on Graphics* 38, 1 (Jan. 2019), 10:1–10:13. https://doi.org/10.1145/3292481

[68] Tianhong Catherine Yu and James McCann. 2020. Coupling Programs and Visualization for Machine Knitting. In *Proceedings of the 5th Annual ACM Symposium on Computational Fabrication (SCF '20)*. Association for Computing Machinery, New York, NY, USA, 1–10. https://doi.org/10.1145/3424630.3425410

[69] Cem Yuksel, Jonathan M. Kaldor, Doug L. James, and Steve Marschner. 2012. Stitch Meshes for Modeling Knitted Clothing with Yarn-Level Detail. *ACM Trans. Graph.* 31, 4, Article 37 (jul 2012), 12 pages. https://doi.org/10.1145/2185520.2185533

[70] Junqiu Zhu, Adrian Jarabo, Carlos Aliaga, Ling-Qi Yan, and Matt Jen-Yuan Chiang. 2023. A Realistic Surface-based Cloth Rendering Model. In *ACM SIGGRAPH 2023 Conference Proceedings* (, Los Angeles, CA, USA,) *(SIGGRAPH '23)*. Association for Computing Machinery, New York, NY, USA, Article 5, 9 pages. https://doi.org/10.1145/3588432.3591554

A KNITTING OVERVIEW

This section provides a high-level overview of the domain of knitting and definitions for terms used in this paper. For a comprehensive text on knitting technology, we refer to Spencer [60].

Knitting refers to the process of manipulating yarn into fabric via interlooping, in which loops of yarn are pulled through other loops. Knitting is often compared (and confused) with *weaving*, where fabric is formed by interlacing two sets of yarns at right angles. Both techniques produce different attributes in the fabric with different applications: knits are typically more breathable and stretchier (T-shirts, socks), while wovens are firm and stable (denim, bedsheets, towels).

Hand knitting is at least 1500 years old and continues to be extremely popular today. Hand knitting is typically performed on straight pointed needles. As a general rule, hand knitting has a far larger possibility space than machine knitting, so anything that can be knit on a machine can be knit by hand. However there are practical tradeoffs in terms of speed, pattern complexity, and consistency.

Knitting machines also use needles, but each needle holds only a small number of loops at a time (typically only one or two). Knitting machine needles are shaped like a small hook with a latch that can open and close. On the machine, needles are arranged into one or more *beds*. In this paper we are concerned with *V-bed knitting machines*, in which two beds are oriented in an upside-down V shape, with the needle hooks facing each other at the top. Other classes of knitting machines have different bed configurations. For example, circular knitting machines have a circular bed.

A *carriage* moves along the needle bed to guide the yarn and actuate the needles according to the pattern. Typically, machines have multiple *yarn feeders* which the carriage will pick up and drop off according to which yarn is in use. During the carriage movement, the type of stitch performed by each needle is determined by how far it moves out and in. To knit a stitch, when the needle moves out the previously-held loop slides back behind the latch, and a new loop of yarn is laid into the open hook. Then, as the needle moves back in, the previous loop slides forward, pushes the latch closed, and drops over the closed latch.

B GLOSSARY OF TERMS

stitch	Used interchangeably to refer either to: a single loop of yarn embedded in fabric, the unit operation which produced it, or to a repeating texture pattern (e.g. "seed stitch").
knit	Refers to fundamental knitting operation in which a loop of yarn is pulled through an existing loop. The yarn can be pulled through a loop from either side, producing either a knit (which resembles a "v") or a purl (resembling a small bump). In machine knitting, these operations are often referred to by the bed used to create them ("front knit" or "back knit"), while hand knitters refer to them as "knits" and "purls".
tuck	An operation where a loop of yarn is added to a needle without pulling it through an existing loop (if present)
slip, skip, miss	An operation where yarn is passed behind a loop without being pulled through it. In machine knitting, the latch hook needle does not extend to pick up a new loop. In hand knitting, the loop is passed without modification from one straight needle to another. In both cases, the extra yarn produces a *float*.
transfer	An operation where loops are moved from one needle to another, often used to create textures like lace or cables or to shape the fabric.
float	A loose bit of yarn not directly worked into the fabric structure. Typically there is an upper limit to the length of a float, as floats which are too long may snag on fingers or jewellery.
chart	A visual representation of a knitting pattern, in which symbols and colors are arranged on a grid to communicate the placement of stitches, yarns, and techniques across the rows and columns of the fabric.
colorwork	An umbrella term describing knitting techniques where different colors of yarn are incorporated into a single piece to create color patterns.
stranded or fair-isle colorwork	A colorwork technique for working two or more colors of yarn in the same row by floating the non-showing yarn across the back of the fabric.
intarsia	A colorwork technique where each separate block of color in a row is knit by a dedicated yarn. This means that yarns are not floated across the back of the work (as they are in stranded colorwork), making it an ideal technique for larger, solid-colored areas.
cable	A texture effect where loops are "crossed", which can create the appearance of twisted rope on the fabric surface. Knitting machines use transfers to rearrange the loops, and hand knitters typically use an extra needle (a "cable needle"). Cable knitting is a family of techniques for producing different effects via crossed stitches, including braids, twists, and lattices.
rib	A texture created by alternating knit and purl stitches in vertical columns. This produces a very stretchy fabric commonly used for cuffs, collars, and hems.
lace	A technique that incorporates patterns of holes or gaps, often created by using transfers to move loops to create holes and form intricate designs.

C PSEUDOCODE FOR BOUNDARY AND PATH RASTERIZATION

Algorithm 1: Boundary Rasterization

```
Function AddEdge($EdgeTable$, $[x1, y1] \leftarrow p1$, $[x2, y2] \leftarrow p2$):
    if $y1 = y2$: return;
    if $y1 > y2$:
        $[x1, x2] \leftarrow [x2, x1]$;
        $[y1, y2] \leftarrow [y2, y1]$;
    $dx \leftarrow (x2 - x1)/(y2 - y1)$;
    $edge \leftarrow$ edge {
        x $\leftarrow x1 + (dx/2)$;
        dx $\leftarrow dx$;
        yMin $\leftarrow y1$;
        yMax $\leftarrow y2$;
        xLast $\leftarrow x1$;
    };
    $EdgeTable$.insert($edge$);
Function RasterizeBoundary($boundary$):
    $points$, $stitchBlock$, $yarnBlock$, $offset$, $shaping \leftarrow boundary$;
    $EdgeTable \leftarrow []$;
    for $i \leftarrow 0$ to $points.length$:
        AddEdge($EdgeTable$, $points[i]$, $points[(i+1)\%points.length]$)
    $EdgeTable \leftarrow EdgeTable$ sorted on $x$;
    $ActiveEdgeTable \leftarrow []$;
    $y \leftarrow 0$;
    while there are edges left to process:
        move edges from $EdgeTable$ to $ActiveEdgeTable$ where $yMin = y$;
        for $i \leftarrow 0; i < ActiveEdgeTable.length; i \leftarrow i + 2$:
            $left \leftarrow$ ActiveEdgeTable[$i$];
            $right \leftarrow$ ActiveEdgeTable[$i + 1$];
            $x0 \leftarrow$ round(left.x);
            $x1 \leftarrow$ round(right.x);
            if $x0 = x1$: continue;
            $StitchBlockY \leftarrow (y$ - $offset.y)$ % $stitchBlock$.height;
            $YarnBlockY \leftarrow (y$ - $offset.y)$ % $yarnBlock$.height;
            for $x \leftarrow x0$ to $x1$:
                $StitchBlockX \leftarrow (x$ - offset.x) % $stitchBlock$.width;
                $YarnBlockX \leftarrow (x$ - offset.x) % $yarnBlock$.width;
                $stitch \leftarrow stitchBlock$.at($StitchBlockX$, $StitchBlockY$);
                $yarn \leftarrow yarnBlock$.at($YarnBlockX$, $YarnBlockY$);
                if $stitch \neq TRANSPARENT$: DrawStitch($stitch, x, y$);
                if $yarn \neq TRANSPARENT$: DrawYarn($yarn, x, y$);
                if $shaping > 0$:
                    ShapeLeft($left.xLast - x1$);
                    ShapeRight($right.xLast - x2$);
                $left.xLast \leftarrow x1$;
                $right.xLast \leftarrow x2$;
        remove edges from $ActiveEdgeTable$ where $yMax = y$;
        foreach edge in ActiveEdgeTable do
            $edge.x \leftarrow edge.x + edge.dx$;
        $y \leftarrow y + 1$;
```

Algorithm 2: Path rasterization

```
Function PlotPath(p0, p1, block, offset, mode):
    $[x0, y0] \leftarrow p0$;
    $[x1, y1] \leftarrow p1$;
    $dx \leftarrow abs(x1 - x0)$;
    $sx \leftarrow x0 < x1 ? 1 : -1$;
    $dy \leftarrow -abs(y1 - y0)$;
    $sy \leftarrow y0 < y1 ? 1 : -1$;
    $error \leftarrow dx + dy$;
    $last \leftarrow [x0, y0]$;
    while True :
        switch mode do
            case “overlap“ do
                DrawBlock($block, x0 + offset.x, y0 + offset.y$)
            case “tiled” do
                if $abs(x0 - last.x) \geq block.width$ or $abs(y0 - last.y) \geq block.height$ :
                    DrawBlock($block, x0 + offset.x, y0 + offset.y$);
                    $last \leftarrow [x0, y0]$;
            case “dx” do
                if $x0 \neq last.x$ :
                    DrawBlock($block, x0 + offset.x, y0 + offset.y$);
                    $last \leftarrow [x0, y0]$;
            case “dy” do
                if $y0 \neq last.y$ :
                    DrawBlock($block, x0 + offset.x, y0 + offset.y$);
                    $last \leftarrow [x0, y0]$;
        if $x0 = x1$ and $y0 = y1$ : break;
        $e2 \leftarrow 2 \times error$;
        if $e2 \geq dy$ :
            if $x0 = x1$ : break;
            $error \leftarrow error + dy$;
            $x0 \leftarrow x0 + sx$;
        if $e2 \leq dx$ :
            if $y0 = y1$ : break;
            $error \leftarrow error + dx$;
            $y0 \leftarrow y0 + sy$;

Function RasterizePath(path):
    $points, stitchBlock, yarnBlock, offset, mode \leftarrow path$;
    for each segment pair $p0, p1$ in points :
        PlotPath(p0, p1, stitchBlock, offset, mode);
        PlotPath(p0, p1, yarnBlock, offset, mode);
```