Computational Illusion Knitting

AMY ZHU, University of Washington, USA
YUXUAN MEI, University of Washington, USA
BENJAMIN JONES, University of Washington, USA
ZACHARY TATLOCK, University of Washington, USA
ADRIANA SCHULZ, University of Washington, USA

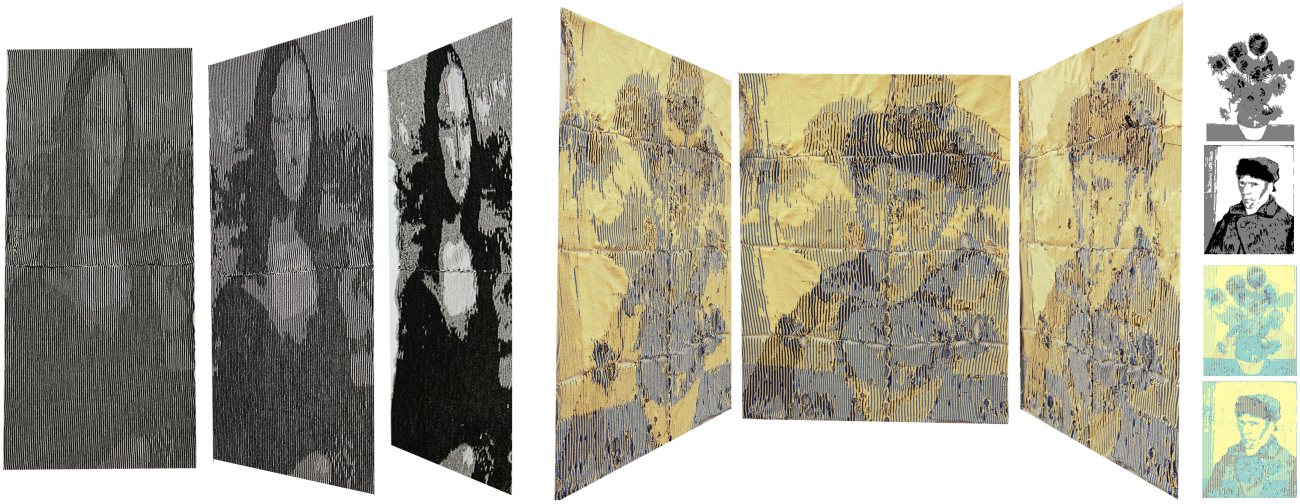


Fig. 1. We propose a framework for computational illusion knitting that enables simple production of traditional and completely novel illusion designs, and show the fabricated designs generated with our method. On the left, Leonardo da Vinci's Mona Lisa (public domain) appears when viewed from the side. On the right, Vincent van Gogh's Self-Portrait with a Bandaged Ear (public domain) appears from one side, his Sunflowers, fourth version (public domain) on the other. The rightmost top two images are the tool inputs; the bottom two are rendered predictions of the illusion pattern.

Illusion-knit fabrics reveal distinct patterns or images depending on the viewing angle. Artists have manually achieved this effect by exploiting "microgeometry," i.e., small differences in stitch heights. However, past work in computational 3D knitting does not model or exploit designs based on stitch height variation. This paper establishes a foundation for exploring illusion knitting in the context of computational design and fabrication. We observe that the design space is highly constrained, elucidate these constraints, and derive strategies for developing effective, machine-knittable illusion patterns. We partially automate these strategies in a new interactive design tool that reduces difficult patterning tasks to familiar image editing tasks. Illusion patterns also uncover new fabrication challenges regarding mixed colorwork and texture; we describe new algorithms for mitigating fabrication failures and ensuring high-quality knit results.

Authors' Contact Information: Amy Zhu, amyzhu@cs.washington.edu, University of Washington, Seattle, Washington, USA; Yuxuan Mei, ym2552@cs.washington.edu, University of Washington, Seattle, Washington, USA; Benjamin Jones, benjones@cs.washington.edu, University of Washington, Seattle, Washington, USA; Zachary Tatlock, ztatlock@cs.washington.edu, University of Washington, Seattle, Washington, USA; Adriana Schulz, adriana@cs.washington.edu, University of Washington, Seattle, Washington, USA.

ACM 1557-7368/2024/7-ART152
https://doi.org/10.1145/3658231

CCS Concepts: • **Computing methodologies** → **Graphics systems and interfaces**; • **Applied computing**;

Additional Key Words and Phrases: illusion knitting, machine knitting, knitting, fabrication

ACM Reference Format:
Amy Zhu, Yuxuan Mei, Benjamin Jones, Zachary Tatlock, and Adriana Schulz. 2024. Computational Illusion Knitting. *ACM Trans. Graph.* 43, 4, Article 152 (July 2024), 13 pages. https://doi.org/10.1145/3658231

1 INTRODUCTION

Unique and visually interesting view-dependent effects have been developed for many fabrication methods and mechanisms, including scratch-based reflection, 3D printing, shadow casting, and more [Alexa and Matusik 2011; Bermano et al. 2012; Shen et al. 2023]. However, there is currently no support for view-dependent effects in knitting. Intrepid artists have described manual, hand-knitted view-dependent effects (dubbed "illusions" [WoollyThoughts 2023]), but their approaches are tedious, slow to knit up, and limited to single-view; it takes hundreds of hours to design a knitted illusion, and hundreds more to actually fabricate the knit object. Though machine knitting is an obvious choice to speed up fabrication, no compilers for illusion knitting currently exist. Nor are there machine knitting scheduling algorithms that support arbitrary intra-row texture and colorwork, necessary for maximizing design freedom. Furthermore, artists have created only single-view illusions, which depict the

same image from both side views, but we imagine enriching the design space by creating novel illusions that embed multiple images.

We create the first design system for illusion knitting, enabling design without cumbersome manual effort, pushing the boundaries into multi-view illusions, and correctly compiling to machine knitting instructions.

First, we identify how illusions work. Illusion knitting emerges from stitch-level *microgeometry* on the surface of knits (depicted in Figure 3); we contribute a set of observations about microgeometry formalized as constraints, such that we can directly construct these building blocks. These constraints capture both *viewing behavior*, i.e., what image is seen at an angle, and *physical behavior*, i.e., the result of different knitted design and fabrication choices.

Formalizing these constraints provides immediate benefits: for single-view, we can easily satisfy the constraints to generate knittable illusions. Furthermore, as traditional single-view illusions change color only between rows, we require no new machine knitting algorithms. Therefore, we can quickly design and computationally fabricate single-view illusions, like the Mona Lisa in Figure 1, given a user-provided three-color image.

Trying to extend these same solutions for single-view to multi-view illusions, we discover that they are inadequate and so we must invent other strategies. Our key contributions focus on new techniques for multi-view illusions.

Unfortunately, the constraints for multiple images are almost never satisfiable. Our insight is that *we can relax these constraints* and search for the best compromise. We could do this using automated methods, such as optimization or MaxSAT. However, we believe it important to begin with a framework that lets users directly, interactively, and iteratively edit designs, since an end-to-end automated system cannot return a perfect result for every user 100% of the time. Because users employ human perception to gauge whether designs are readable, we create a *designer-in-the-loop framework* that uses insights from craft illusion knitting to simplify the task of improving illusion quality to iterative image editing. This framework relies on a new tile-based system to systematically relax constraints while giving users direct control. Creating a framework also preserves extensibility as we can plug in other automated algorithms.

We then face the issue of new designs intermingling texture and colorwork within rows. Given this task, existing machine scheduling algorithms fail to create objects without unworkable physical flaws. To fabricate knit illusions quickly, we introduce *new scheduling algorithms for texture and color changes within a single row of knitting*, which is necessary for multi-view illusions.

Our constraint-informed, human-in-the-loop design system enables fabricating the first double-view illusions, supporting diverse image types across graphics, text, paintings, and more. Additionally, our new scheduling approach avoids many failures that arise with naive scheduling, producing successful knit illusions.

As evidence, we present multiple complex knit illusions never previously created, examples of which we show in the teaser (Figure 1). We compare the output of our iterative, image-editing design system with other systems. Finally, we visually compare the fabricated results of knitting with and without our new scheduling algorithms, where the latter have notable holes and visual failures.

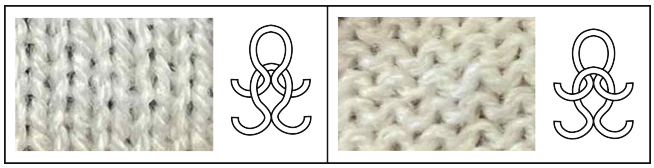


Fig. 2. A basic view of knits and purls. On the left is a photo of a fabric made of only knits next to a diagram of a single knit; on the right is a photo of a fabric made only of purls next to a diagram of a single purl.

2 BACKGROUND

Knitting Terminology. A *stitch* is the basic knit unit formed by pulling a loop of yarn through an existing loop. We refer to the existing loop as the "parent" and the new loop formed as the "child". Rows of loops are called *courses*, and columns of loops are called *wales*. The direction in which the yarn is being pulled through each loop determines the stitch type, resulting in knit or purl stitches. *Knit stitches* are created when the new loop formed is "in front of" the parent loop (pulled back-to-front), and *purl stitches* are created when the new loop is "behind" the parent loop (pulled front-to-back). These stitches are visualized in Figure 2. There are many different *textures*: arrangements of knits, purls, or other types of stitches. For example, the garter texture alternates courses of knits and purls; the rib texture alternates wales of knits and purls. Full specifications of knit designs are called *patterns*.

Several factors affect the appearance of knit objects. Knits and purls are *duals*, meaning that a purl viewed from the back is a knit, and vice versa. Thus, knit fabrics often have a "right side,", or a "technical face," from which the pattern is designed to be viewed. In this work, we focus on the appeal of the technical face and allow the back to be aesthetically unconstrained. Additionally, there is a metric consideration since the size of a knitted object is directly related to the arrangement and size of the loops. The *stitch gauge* describes the number of stitches per unit distance. In hand knitting, the gauge is controlled by the size of the needle used for creating the loops and the size of the yarn, while in machine knitting it is determined by needle size and how close the needles are arranged on the beds.

Illusions. *Illusion knitting* is a technique used to create knit objects that appear different when viewed at different angles. It exploits occlusions between neighboring courses created by particular arrangements of knits and purls. Traditional illusion knitting achieves a single hidden image in each piece by disguising the occlusion effect within alternating color stripes. All known single-view techniques use only two yarn colors in the illusion effect, but it is possible to create the impression of a third color by keeping both colors visible (e.g., raised black next to raised white creates the impression of gray). Section 5 further describes how to achieve single-view illusions.

3 RELATED WORK

Fabricated View-Dependent Effects. View-dependent effects play on our visual perception via lighting and shadows, utilizing self-occlusion or self-shadowing to change the visibility of parts of the object; these objects then reveal different images at different viewing

(a) A top-down look at a single-view pattern. The knitted version is depicted in Figure 4b.

(b) A close up of the microgeometry of single-view patterns. At the bottom is a row of yellow purl bumps, below blue and yellow knits. The yellow purl bumps are raised above the fabric.

Fig. 3. Physical view of illusion patterns and microgeometry.

angles. Some works achieve a specific appearance of a surface under a certain lighting condition or environment through optimizing microfacets [Papas et al. 2011; Regg et al. 2010; Schwartzburg et al. 2014; Snelgrove et al. 2013; Weyrich et al. 2009] or lenticular structures [Zeng et al. 2021]. Many fabrication techniques and materials have been employed, including Lego bricks [Mitra and Pauly 2009], 3D printing [Alexa and Matusik 2011, 2012; Bermano et al. 2012; Peng et al. 2019], UV printers [Perroni-Scharf and Rusinkiewicz 2023; Sakurai et al. 2018], reflection via scratches on metal [Shen et al. 2023], and printing with magnetic [Pereira et al. 2017] or goniochromatic [Abu Rmaileh and Brunton 2023] materials. Also a kind of view-dependent effect, illusion knitting relies on precisely occluding certain stitches – the microgeometry in a knitted object – to embed images in different viewing angles.

Computational Knitting. Several recent papers explore computational tools for knitting design and fabrication. Some focus on how to represent knit objects for simulation [Wu et al. 2018; Yuksel et al. 2012]. Albaugh et al. [2023] also study the grain of machine-knit fabric. Others focus on creating knitting instructions given a 3D mesh input for both hand knitting [Igarashi et al. 2008; Wu et al. 2019] and machine knitting [Jones et al. 2022; Kaspar et al. 2021; McCann et al. 2016; Narayanan et al. 2018; Popescu et al. 2018]. Further work builds upon these pipelines to improve knitted results, such as new striping algorithms to avoid helices [Mitra et al. 2023]. Nader et al. [2021] focus on a machine-agnostic, hardware-independent intermediate representation. Design methods have been developed for specific applications, including actuated soft objects, pneumatic devices, and conductive interfaces and sensing devices [Albaugh et al. 2019, 2021; Liu et al. 2021; Luo et al. 2021, 2022]. Texture design is another important topic of research. Narayanan et al. [2019] provide a tool that allows easy stitch-level editing of textures and colorwork while providing machine knittability guarantees. Hofmann et al. [2020] examine knit textures and provide a DSL for describing them.

Although colorwork is a well-known facet of computational knitting, most existing work [Lin and McCann 2021; Lin et al. 2023, 2018; McCann et al. 2016; Narayanan et al. 2019] provides insights only on how to schedule patterns with either colorwork or texture. Recently, Hofmann et al. [2023] proposed KnitScript, which provides float control and a primitve called *sheets* for mixing colorwork and texture; it does not focus on this mixed design space, and the float control must be manually specified by the user. Therefore, no past work has focused on exploring and enabling the design space spanning a combination of colorwork and texture, which is a necessary component of illusion knitting and a potential new tool in the democratized designer's toolkit.

Illusion Knitting. Illusion knitting has been explored by knitters for several decades, particularly in garments [Harmon 2011; Hoxbro 2004; Nakamura et al. 1982; Stoller 2004] and art exhibition pieces [WoollyThoughts 2023]. Current illusion knitting artwork typically hides a single image in the side view within an apparent striped image. In terms of design, Orochi et al. [2006] host a tool for hand-charting illusion patterns. However, little work extends the typical single-view-image-in-stripes idea. Some works include patterns that change colors between different areas [Hoxbro 2004; Nakamura et al. 1982], but they use the same fundamental two-color illusion strategy. No resources discuss multiple feature images in one knit illusion. In this paper, we establish the foundation for understanding the broader space of illusion knitting and develop methods for hiding more than one image, using arbitrary colors, and mitigating fabrication challenges in using knitting machines.

4 OVERVIEW

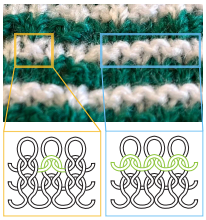
Single-view illusions show that exploiting knit microgeometry can achieve surprising designs, and we intend to push the boundaries to discover even more possibilities using computation. In this work, we consider illusions that embed **two** input images, A and B, where image A is obvious from one side and image B from the other side. However, we design approaches that can theoretically extend beyond two inputs. Additionally, since the landscape of knitted microgeometry is very diverse, we make it navigable by restricting the discussion to illusions created via solid-color microgeometry with two possible heights (flat and raised).

To create a design system, we must first identify what is possible in terms of the appearance of knitted microgeometry when viewed. Doing so involves knowledge of fabrication as well as physical experimentation. We draw on these insights to develop new, constrained design spaces using logical constraints and a new design system for traversing them.

4.1 Insights from Knit Microgeometry

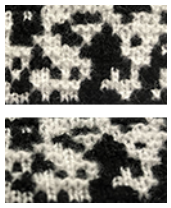
To characterize a knit object's microgeometry, we examine empirical examples of knit illusions. Our goal is to abstract over the complexities of knitting physics so we can streamline illusion design.

Combinations of knit and purl stitches create raised *bumps* that may occlude previous stitches. A bump is created from a knit with another loop pulled through it front-to-back; in other words, a bump is made of a knit with a purl as a child. Not every combination creates a crisp occlusion effect, and we present some mistakes in Figure 4. First, a single bump is not sufficiently tall to occlude a stitch behind it from almost all angles. The legs of a bump flanked by two flat stitches are pulled down, shown in Figure 4a. Knit stitches alone remain *flat*. Second, directionality matters. Theoretically, we could view and produce bumps along wales rather than courses, working in ribbing rather than garter stitch. However, we find that height differences are more pronounced in garter, as shown in Figure 4b. Third, different programs and techniques might produce excess bulk, whether in the form of extra "floating" yarn at the back, doubling up the yarn used per stitch, or via other techniques. This bulk reduces



(a) The height of a 1-wide purl bump is greatly diminished when its legs are pulled under the next stitch. When next to other purls, the legs stand above the fabric.

(b) An illusion pattern: garter on top (colors switching each course) or rib on bottom (switching each wale). Frequent color changes in the attempted rib illusion add too much bulk.



(c) Frequent color changes diminish occlusion effects. The top image is the head-on pattern. Rows of purl bumps do not obscure many of the stitches behind them in an angle view, bottom.

Fig. 4. Some pattern choices negatively affect occlusion. Without carefully arranging stitches, occlusion effects do not appear.

the height difference between flats and bumps, diminishing the illusion effect. We can avoid some techniques that accrue extra bulk, but not all: each color change within a row inevitably incurs some added bulk.

From these insights, we draw three conclusions. From the first, we choose to introduce an abstraction over these physical constraints. In this *logical* space, flat and bump *units* are individually compiled to stitches such that the bumps are guaranteed to occlude other stitches at an angle, i.e., each pixel becomes a 2x2 block of stitches. From the next insight, we define our objective to view the first input image from the left side of the rows and the second input image from the right side of rows. From the final insight, we conclude that our designs should mindfully reduce the number of color changes.

4.2 Defining a Design Space

To identify which double-view illusion designs can be knit "perfectly" in principle (ignoring practical fabrication concerns), we formalize the constraints for when $N \times M$ pixel images A and B will be obvious within the result of knitting the logical (color and bump level) pattern P, where $C_{i,j}$ and $R_{i,j}$ indicate the color and raisedness of P at row i and column j, respectively. Since the first row will not be occluded by any previous rows for A, and similarly for B with the last row, we require $C_{0,j} = A_{0,j}$ and $C_{N,j} = B_{N,j}$. For subsequent rows, we have $\neg R_{i-1,j} \implies C_{i,j} = A_{i,j}$ and $\neg R_{i+1,j} \implies C_{i,j} = B_{i,j}$ (i.e., if the cell in the previous column is flat, then our current cell will be visible and must match the color of the corresponding cell in the input image), as well as $R_{i-1,j} \implies C_{i-1,j} = A_{i,j}$ and $R_{i+1,j} \implies C_{i+1,j} = B_{i,j}$ (i.e., if the cell in the previous column is raised, then it will occlude our current cell, so the cell in the previous column's color must match our current corresponding cell in the input image). These constraints implicitly specify visual rendering restricted to booleans; we present a full rendering function in Section 5.

The boolean rendering highlights the fact that multi-view illusion knitting is over-constrained: for the vast majority of pairs of images, no pattern can satisfy all the requirements. Intuitively, without bump units, our images cannot differ since bump units occlude the units behind them and create the illusion. However, *the color of every bump unit must match both images* since it will be visible from both sides.

We can attempt to minimize error using automated techniques, but human perception ultimately decides illusion quality. Therefore, we establish a first human-in-the-loop framework that can support both automated design as well as interactive, iterative editing.

To do so, we abstract over the logical space to create an editing space for designers. This design space enables robust approaches to generating obvious and knittable illusions by *reducing the development of illusion patterns to image editing*.

Consider the pair of inputs from the inset figure. These inputs are unsolvable: one image demands white, where the other demands black. How can we create a double-view illusion at all?

To begin, we observe that using the logical constraints specified previously, it is possible to have the left and right views differ. Consider a 3 by 1 repeating tile, i.e., white flat, black flat, and white bump. From the right, the white bump occludes the black flat, and the image appears white. From the left, we see the white bump **and** black flat, creating a black and white striped "fill." Incidentally, we notice that in craft illusion knitting, simple repeating patterns are occasionally used as fills; for example, a striped black-and-white pattern looks like a medium gray against solid black and solid white. Leveraging the idea of black/white stripes as a fill, we can susbtitute the solid black in both inset inputs with a black/white stripe to make this illusion possible using our new tile, as shown in Figure 5.

This approach provides yet another new insight; we can modify input images to make robust double-view illusions possible. In other words, we must relax the illusion constraints. Doing so requires striking a balance between three factors: (1) constraint satisfaction (i.e., does the pattern actually replicate the input image?), (2) knittability (i.e., is it actually feasible to create the pattern?), and (3) readability (i.e., if we modify the input to improve satisfiability, will the modified input still be perceived as similar to the original input?). For example, relaxing all inputs into a white square would make them knittable and constraint satisfying but not readable as most input images. Or, prioritizing satisfiability, we could use an approach such as MaxSAT or optimization. However, using these relaxation methods might make it more difficult to control knittability and readability, as shown in subsection 8.3.

To successfully navigate this tradeoff, we can draw on some observations from craft illusion knitting. Knit illusions become less readable when fabricated. With a pixel mapped to 2x2 stitches, the resulting knit objects are large, meaning input images are typically low-resolution. As cloth, the fabric is wavy, so details are often swallowed up. In response, artists segment images into large swaths of each color and avoid dithering, instead employing regular, repetitive patterns to make the borders between colors distinct. We must relax the input simply and *consistently* to preserve readability. This relaxation should be editable so the user has control.

A tile-based system balances our three concerns. Repainting input regions with these new fills makes the constraints within each region satisfiable since the fills are backed by tiles. Tiles are designed to be fabrication-sensitive; our chosen patterns create unbroken courses

Fig. 5. A knit version of our recolored input patterns. (Left) The black background and white square have become a striped background and white square. (Right) The white background and black square have become a white background and striped square.

of color within each tiled region. This property results in shorter fabrication times with less complexity. Tiles also consistently relax the input; all instances of the same colored regions are repainted with a color or a simple, regular fill, maintaining readability. Best of all, relaxation is controlled by the user when they define the color regions of the input and decide how to repaint them. Users can fine-tune elements by editing the quantized design, reducing the problem to image editing, which can be performed in any familiar software.

Thus, we propose a *tile-based framework* for creating illusions. We provide a *tile bank* of template tiles that have known fills from each view, and which can be customized and extended. The user provides simplified and quantized input images. The framework then overlays the two inputs onto each other, recording the original color specification. It then repaints the original colors in the image with new fills, drawing from the tile bank for the matching tiles (Figure 6). Crucially, multiple tiles might map to the same fill on one side but to a different fill on the other side. The tool then displays the predicted knitted output, and users are free to edit what tiles are used.

In this way, editing becomes as simple as modifying the input image regions with a preferred image editor, knowing that knittability and satisfiability will be preserved.

5 FOUNDATIONS OF ILLUSION KNITTING

To reason about correct illusions, we seek to establish a formal understanding of illusion knitting. We start by describing a function called `can_see`, which mathematically identifies visible logical units in the side view. The definition of `can_see` is given by the constraints defined in subsection 4.2, such that `can_see` from one direction is given by the constraints for A and from the other direction given by the constraints for B. Using `can_see`, we define a "perfect" illusion as one where the pattern viewed from the side exactly matches the input image.

Constraints for Single-View Illusions. We use `can_see` to write an algorithm for generating single-view illusions from a two-color input. Our algorithm, shown in Figure 7, is heavily inspired by craft examples. We alternate striped rows of colors 1 and 2 ($c1$ and $c2$). However, we translate only $c1$ pixels to bump units if the stripe row is $c1$-colored, while $c2$ pixels in $c1$ rows are translated to $c1.flat$ units. This algorithm produces SAT patterns (modulo boundary rows). While we ignore $c2$ pixels in $c1$ rows, we raise the stitches only where $c1$ is used. So, a $c1.flat$ will never block a $c2.flat$ in the next row. Given that all pixels are either $c1$ or $c2$, $c2$ pixels in row n (when n is a $c1$ row) will appear in $n+1$ in the knit result.

For inputs with an intermediate third color, we can extend this algorithm to translate $c3$ to two adjacent bump rows. This modification maintains approximate correctness.

Accelerating design for single-view illusions is simpler than our goal of doing so for double-view, but we are still content that our push-button solution for single-view can replace hundreds of hours of work, lowering the barrier to entry for designing such illusions.

Differentiable Renderer. Though we have a formal renderer defined via our constraint system, we additionally observe that the angle at which the image is viewed affects perception. Providing a more flexible, accurate view is essential for interactively exploring the design space. In real life, viewers will naturally move around to find the sharpest image. Our new ray-casting renderer is differentiable, enabling optimizations to be built on top of it.

Two simplifying assumptions allow for autodifferentiating stitch occlusion. First, we treat logical stitches as cuboids of fixed height. Second, we fix a camera that is orthographic with respect to the width. As a result, all predictions are rectangular, which aids when fitting an illusion knit pattern to the target input. Thus, viewing rays can only intersect with the leading or top faces of each stitch cuboid within one wale. The 3D rendering problem is thereby reduced to a single 2D problem with wales computed independently.

To render a wale, we analytically compute the contribution of each line segment S_i to each pixel P_i, accounting for occlusion by earlier segments $S_{j<i}$: $|\left(S_i \setminus \cup_{j<i}\left(S_j\right)\right) \cap P_i|$. Here, S_k and P_k are the projections of the $k-$th stitch and pixel onto image space. We show a differentiable closed form expression in the Supplemental Material.

To tune the renderer's parameters, we knit a test pattern, shown in Figure 8. We measured the angle at which each purl bump configuration obscured the opposite-color stitches and used this to calculate the height of the purl stitches. The same pattern can be used to tune parameters for a new setup, e.g., when using a thicker yarn or different gauge needles. We show the results of rendered outputs compared to knit outputs in Figure 18. The rendered output strongly replicates the real, physical illusion effect, successfully making immediate visual feedback possible.

6 DESIGN FRAMEWORK

Double-view illusions are difficult because they are over-constrained. In fact, every example in our evaluation was not satisfiable in the original input. Thus, we propose a design system based on *tiles*, which relax the illusion constraints in a constraint-satisfying, knittable, and readable way. Using tiles as a fundamental construct, we reduce illusion design tasks to familiar image editing tasks, e.g., how to quantize input images into large colored regions, without requiring knitting knowledge. Combined with our differentiable renderer for previewing illusions without actually manufacturing them, our approach simplifies and accelerates iteration. We now describe in more detail the development of this framework.

6.1 Developing Tiles

Guided by the illusion constraints, we design a series of new canonical tile types, pictured in Figure 10. A tile maps to a fill of some color (solid or two-color alternating stripes) and can be used to design single-view head-on and double-view illusions, or more. We chose only simple striped fills since they are the common non-solid fill used in artists' illusion knits. Along with the principle that a bump can block one unit beyond it, we naturally ended up with a series of

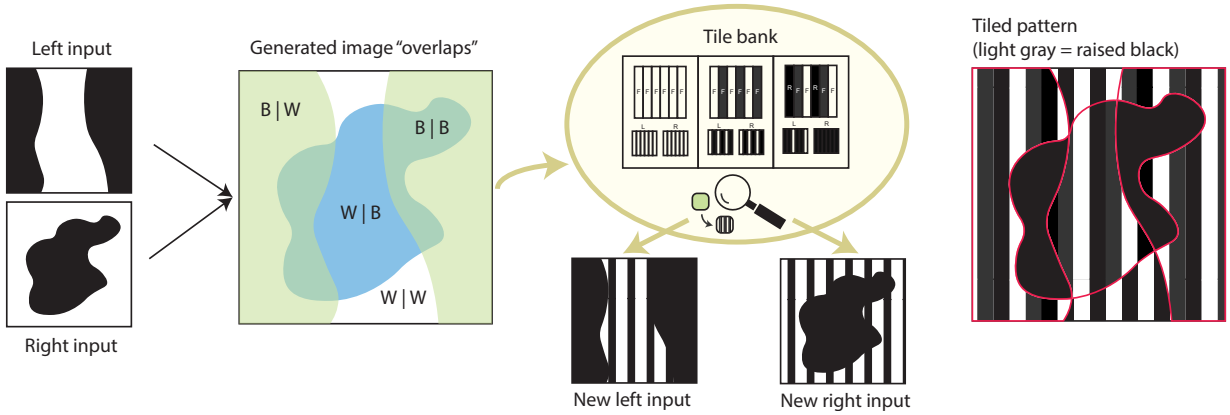
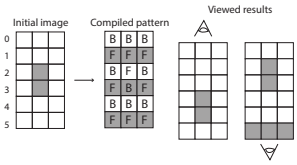
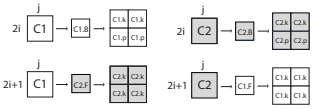


Fig. 6. The workflow for our tile-editing framework. (Leftmost) Our original inputs for each view, with large, simple color regions. (Center left) The system then overlaps the images, creating many new smaller regions where the two inputs had different colors. For example, the blue region in the middle should be white on the left and black on the right. (Center right) The system repaints the inputs with new fills. It must assign a new fill to each color c in the original left input such that for each overlap region $c|d$, there is a tile that shows c on one side and d on the other, and same for the right. (Rightmost) The final tiling.



(a) An example of our compilation algorithm and the result of can_see. The two rightmost images are the viewed image from the top and bottom, respectively, according to can_see.



(b) A visual description of our single-view image-to-illusion compilation algorithm. Each pixel in the input is compiled first to a logical unit and then to a physical unit depending on its row of origin.

Fig. 7. Illustrations of the single-view compilation algorithm.

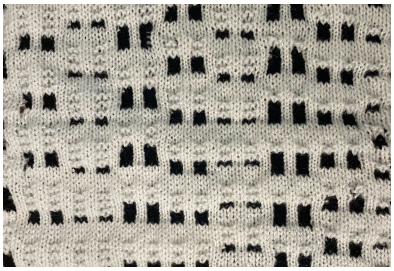


Fig. 8. Our calibration pattern. We knit sets of various sizes of white bumps next to various sizes of black flat stitches. Our examples use different colored yarns of the same dimensions; however, given varied yarn sizes, we could knit a different version of the calibration pattern, with each distinct yarn replacing the white stitches, to learn the height of purl bumps for each yarn.

1 to 3 color tiles with alternating stripes. Variations are possible; e.g., a tile could use a checkerboard pattern instead of straight rows, or could leverage a combination of colors that would look blended from afar, employing more than three colours. Our chosen tile types are easily fabricated yet still flexible and expressive, but motivated users may implement their own tiles custom-designed for other effects.

6.2 Implementing the Image Editing Illusion Design System

Our tool's other main algorithmic function is to suggest mappings from regions to tiles. First, the tool computes the overlap of both inputs. It then attempts to match each original input color to a new fill. In doing so, it must ensure that the overlaps of two new fills is possible — a black and white stripe fill on the right input can overlap with black on the left, but pure white cannot overlap with pure black. Basically, if the two fills share at least one yarn color for all overlapped regions, it is viable. If an assignment for all colors exists, we perform a second filtering check to ensure that the brightness order of the original image is preserved, with stripes being an average brightness of the yarns used.

A default option is automatically applied, so if there are any satisfying assignments, the user already has a quality illusion pattern. To go further, users can select and preview many candidate tiles for each color region. We depict the tool in Figure 9.

6.3 Workflow Example

We demonstrate the power of image editing for generating and improving illusion patterns. For our double-view evaluation example of Van Gogh's Self-Portrait and Sunflowers, we edited the Self-Portrait input (pictured in Figure 11a). Starting with a basic quantization of the input image (Figure 11b), we get the projected result in Figure 12a. The painting in the background of the self-portrait has become unreadable, so we modify the original image by erasing the painting around his head. We also make the bandage clearer by adding a shadow. The resulting image is shown in 11c. These edits can be performed in minutes using a program like Microsoft Paint. The illusion produced from the edits is shown in Figure 12b, and the final knitted image in Figure 1.

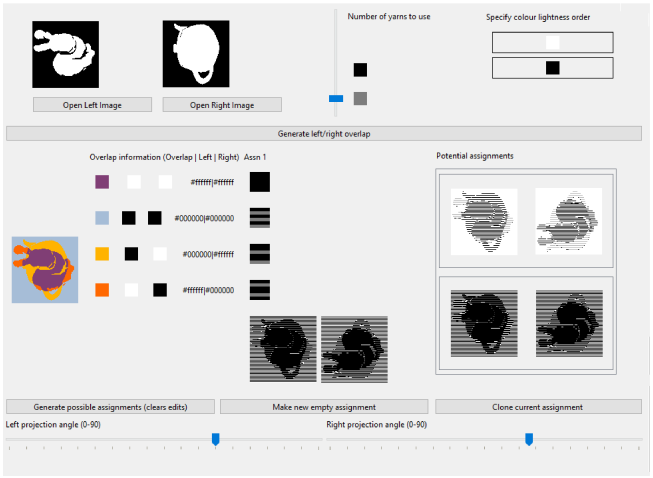


Fig. 9. An illustration of the interface to our simple design system. The interface itself functions as an alternative to running scripts in the command-line and is not meant to be evaluated on its novel interaction. Within the interface, users load their input images, select the number of yarns to use, and press a button to get candidate tile mappings. The rendered output of each mapping is displayed so that the choice is a personal matter of human perception. In many cases, the default tile mapping is the best choice, and users can save the illusion pattern immediately. They may also select another candidate tile mapping that they prefer, again based on the rendered output. Power users may even modify the tiles directly if they have a deliberate effect in mind.

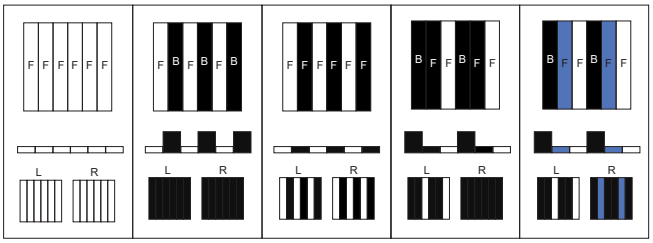


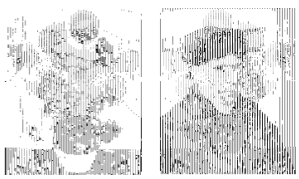
Fig. 10. New tile types (colors can be freely assigned). Tile patterns are shown top-down (top), with its height map (middle) and two side views (bottom).

(a) The original painting as input.

(b) The input image after quantization.

(c) The final cleaned image from user edits.

Fig. 11. The evolution of Van Gogh's Self-Portrait as the designer performs simple image editing and transformation tasks to change the illusion knit output. The final image is the simplest and cleanest, without substantially altering the painting's main details. The edit from (b) to (c) took less than five minutes in an image editing application such as Microsoft Paint.



(a) The rendered result of making an illusion image from Figure 11b and Van Gogh's Sunflowers

(b) The rendered result of making an illusion image from Figure 11c and Van Gogh's Sunflowers.

Fig. 12. Comparing the results of the illusions generated from the originally quantized input image (left) and the edited input image (right). Each set of images is the left- and right-view rendered prediction of the generated pattern. Notice that in the edited image, removing the painting next to Van Gogh simplifies the superimposed overlap, making the rightmost sunflower detail much clearer and the head's definition more distinct.

7 FABRICATION

The illusion patterns we designed require changes between knits and purls and changes between colours within a single row. In hand knitting, the knitter is entrusted to make decisions handling yarn behaviours, e.g., whether the yarn runs along the front or back of the knit object or whether to twist the yarns around each other when they need to catch. In machine knitting, these subtle details must be concretized and managed. Furthermore, operations that non-experts might expect to freely compose, such as knitting a section in one yarn and a second section with another yarn, do not do so because of the implicit changes in machine state that are not explicitly surfaced to the level of abstraction we use to program the machine. Thus, to automatically compile our designs to knitting programs, we must develop new mixed-colorwork-and-texture-aware scheduling strategies to ensure correctness.

7.1 Machine Knitting Background

As opposed to hand knitting, which uses two long needles that holds multiple loops, machine knitting uses a set of adjacent hooked needles in a row (called a bed), each of which hold a single free loop. Each needle then forms a new loop by pulling yarn through the loop it is holding. A *carriage* runs back and forth along the length of the needle beds and actuates each needle. In general, the time it takes to knit an object is a function of the number of carriage passes required.

We focus on enabling fabrication on a V-bed knitting machine, which steeple two opposing beds of needles in an inverted "V", hooks at the top. When knitting texture, the front bed produces knit stitches, the back bed purls. *Carriers* run along rails above the bed to guide yarn over the needles. We have multiple yarn carriers for colorwork but must coordinate them to ensure the knit object stays together. These machines are fully computerized and execute programs written by domain experts. Figure 16 depicts a simplified view of the knitting machine.

7.2 Colorwork Background

Single-view knits posed no scheduling issues. With tiles, however, we must account for color changes within a row, which entails mixing colorwork and texture. Existing machine scheduler algorithms [Lin and McCann 2021; Lin et al. 2023, 2018; McCann et al.

Fig. 13. Examples of colorwork techniques, with the front face (top row) and back face (bottom row). *Doubleknit jacquard* embeds a different image on each side. *Stranded knitting* results in long, unknitted yarn floats across the back. *Intarsia* is seamless from the front, but the yarn- catching used to keep the two different-colored fabrics together is visible along the vertical join on the back. *Plated* produces the inverse of the front design on the back.

2016; Narayanan et al. 2019] are not designed to support both switching between knits and purls and switching colours within a row, as is needed in illusion patterns. One work [Albaugh et al. 2023] deals with the mixed colorwork and texture required for *brioche knitting*, a well-known special style of knitting that uses a particular rib texture and usually employs a colour change to highlight it; we still need new strategies for the arbitrary mixed colorwork and knit-purl texture demanded in illusion knitting.

There are many existing techniques for colorwork. *Plating* works all yarns together, producing color patterns by switching which yarn sits in front. *Stranded* knitting passes unused colors behind stitches, forming strands (a.k.a. floats) along the back. *Intarsia* portions the knitted item into single-color blocks, intertwining the yarns at the borders of the blocks. *Doubleknit jacquard*, arguably the most common technique for machine-knit colorwork, uses all needles on both beds. Importantly, these techniques do not systematically integrate with textures. We now provide a more in-depth explanation of each colorwork technique and illustrate each in Figure 13.

Stranded Colorwork. In stranded colorwork, the knitter transports all yarns as they knit, even if they are not actively in use. Yarns that are not being used for the current loop lie along the back. These loose strands, or *floats*, end when that color's yarn is returned to use, but overly long strands are undesirable. To avoid this, stranded designs employ small motifs close together to keep color changes frequent. Thus, stranded colorwork is appropriate for regions with frequent color changes, but it incurs unnecessary material cost and complexity when a carrier is crossing large swaths of other colors. Also, machine knitting stranded colorwork when patterns have purl textures requires special handling because a naive program will create floats in front of any purl stitches on the back bed.

Intarsia. Instead of carrying every yarn when it is not in use, intarsia knitting divides up regions of colour such that each region is knit with just one yarn moving back and forth, Sometimes, due to how the colour regions are arranged, the project may require two or more instances of one yarn color. Intarsia produces fabric that matches the drape and feel of a single-color knit, but because it requires a new yarn for each color change in a course, it is meant for designs that have larger color regions and few color changes. In general, carrier positions diverge for intarsia, i.e., carriers may move in different directions and be in different ranges of the needle bed. As a result, carriers may not be appropriately positioned to knit the next course and might require extra instructions to ensure correctness. In addition, knitting each section with a different yarn produces disconnected pieces that require explicit joining.

Plated Knitting. In plating, each loop is made with two yarns pulled through the parent loop. On the machine, we can control the appearance of a plated stitch: whichever carrier is "first" contributes most of the color, with the second carrier barely peeking through. Colorwork pieces can be constructed entirely from plating; however, we avoid it because the thickness of plated stitches interferes with the illusion, making knits much taller than non-plated knit stitches.

Doubleknit Jacquard. Jacquard knitting is a very common machine knitting colorwork technique that creates a double-sided stockinette. Notably, the color pattern on the front may completely differ from that on the back. Both the front and back beds are used (and in all-needle jacquard, which is the standard for machine knitting, all needles are used) and the two sides of the fabric are held together when the yarn crosses from one to the other. Because both beds are used, it is impossible to use the back bed needles for purls or transfers at full gauge, which means texture is not easily accomplished.

7.3 Combined Colourwork Strategy

We propose a novel colorwork strategy that supports the colorwork and textures found in illusion knits, based on four key properties of illusion patterns: (1) bumps must be clear so that the illusion is visible; (2) patterns often have many large regions or long runs of a single color; (3) there may be areas with dense color switching, perhaps if there are many small regions; and (4) purls may occur anywhere, and for the knitting machine, where purls are created on the back bed, this means that the back bed cannot freely be used.

From (1), we decided against plating for colorwork since bulky knit stitches interfere with the illusion. Considering properties (2) and (3), neither stranded nor intarsia obviously prevails since one is better suited for frequent and the other infrequent color changes, respectively. We therefore propose a best-of-both-worlds combination of stranded colorwork and intarsia, but we must still address technique-specific challenges. In stranded, floats should not appear on the front of the knit item; in intarsia, different yarns must be securely connected so the piece does not unravel. While solutions exist in typical contexts, they may not apply when combining these techniques or under the constraints imposed by (4). Thus, we introduce three compiler passes to address each concern.

Float Avoidance. Floats appear when a carrier knits on one part of the bed, then moves to knit on another section of the bed. Because the yarn is not cut, it runs loosely along the work. Since carriers move in the space between the front and back beds, floats appear on the front of the work when this movement passes in front of stitches on the back bed. To avoid floats, whenever a carrier moves freely (i.e., not while knitting), we check for purls it might cross in front of, and surround the carrier movement with extra transfer instructions such that stitches move to the front bed before the carrier passes and return to the back bed after. This operation incurs two extra carriage

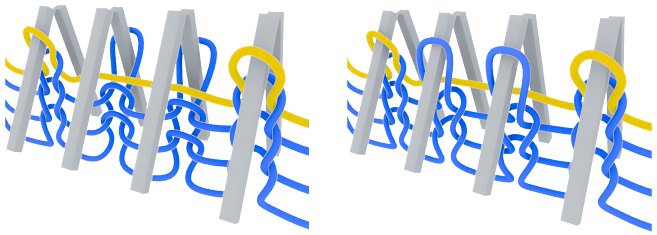


Fig. 14. We depict a 3/4 view representation of the needle bed, with the front and back needles tented inwards and the front needles closer to the camera. The loops currently on the needles are the most recently created, with the fabric moving down as rows are knit. Left: (Incorrect) Floats passing across loops formed on the back bed appear on the front of the fabric. Right: (Correct) Floats passing across the bed go behind loops on the front bed, so they appear on the back of the fabric.

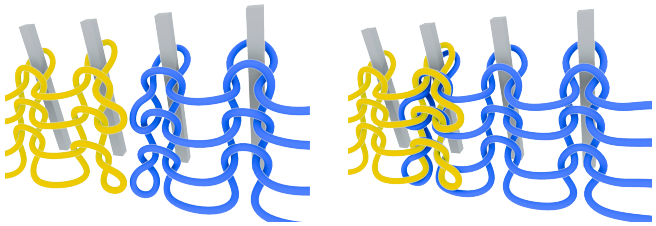


Fig. 15. We depict a 3/4 view representation of the back needle bed, with the front bed omitted. Left: (Incorrect) Knitting two regions of separate colors back and forth without intervention leads to disconnected pieces of fabric. Right: (Correct) Using plating to knit both colors together at the color change ensures that the two halves are intertwined, with the yarn on top contributing the most to the overall colour of the plated stitches.

passes per free carrier movement if there are back-bed stitches to transfer, so checking that this operation is needed saves time.

Algorithm 1 Decide to move purls to the front bed to avoid floats.

```
function FloatAvoidantTransfersGenerator(colorCourse,
carriageDir, backBedNeedlesWithLoops, anchors, program)
    carrier ← carrier for loops in colorCourse
    anchor ← anchors[carrier]
    firstNeedle ← first needle of colorCourse
    pretransfers ← []
    posttransfers ← []
    for needle between anchor and firstNeedle in carriageDir do
        if needle in backBedNeedlesWithLoops then
            pretransfers += xfer(needle, FrontBed(needle))
            posttransfers += xfer(FrontBed(needle), needle)
    end for
    program += pretransfers
    for loop in colorCourse do
        program += getInstsToKnitLoop(loop, carriageDir, carrier)
    end for
    program += posttransfers
    anchors[carrier] ← last loop in colorCourse
end function
```

Yarn Catching. Changing colors typically means using a new yarn, and without explicitly ensuring that two yarns are "caught" (meaning entangled or twisted together), doing so causes gaps in the knit. These gaps are especially apparent when they span multiple rows in the same column. Imagine a degenerate example where a knit piece consists of two different-color rectangles side-by-side. Without catching, the two yarns never contact each other, and the piece is therefore simply two disjoint single-color rectangles.

Yarns do not "catch" on their own since the carrier rail of the knitting machine is designed to avoid yarns tangling. There is a folk method that creates a catch using the back bed. Unfortunately, in illusion knitting, the back bed is densely occupied.

Our strategy uses a plating stitch at the beginning of a color change to catch. Though using plating throughout causes overly thick fabric, we have observed that sporadic use is fine. Both yarns are used in the same stitch and are pulled through one loop together, durably ensuring yarn contact and connecting the two sides. We control which color is more prominent by laying it down first.

Algorithm 2 Modify loop data structure to plate at color changes.

```
function AddPlatingToCatch(loopNeighborPairs, loops)
    currentYarn ← loops[0].yarns[0] ▷ First yarn is "main" yarn
    for n1, n2 in loopNeighborPairs do
        if n1, n2 have different yarns then
            n2.yarns += currentYarn
            currentYarn ← n2.yarns[0]
    end for
end function
```

Yarn Path Planning and Anchoring. In our programs, carriers do not always move together, so some scheduling assumptions do not hold. Given a carriage direction, the knitting machine expects the carrier to originate from behind the carriage in that direction. Consider the case where a carrier is anchored, i.e., has last knit in the middle of the bed, and, in the next course, is going to be used to knit from the very left side to the very right side of the bed. Regardless of which side the carriage starts from, the carrier will originate from in front of the carriage, so no yarn is placed onto the needles and no stitch is formed. This problem is illustrated in Figure 16.

To ensure that every stitch is knit, we propose two different strategies. In the first, we divide the course where the yarn is anchored, first knitting in one direction from the anchor to one side, then turning around and knitting from the anchor to the other side. However, imagine a plated stitch where the carriers involved are on separate sides of the carriage. There is no direction that the plated stitch can be knit such that both carriers will come from behind the carriage. In this scenario, we introduce an *anchoring stitch*. The secondary plating carrier first makes an anchoring stitch (for example, a knit) to anchor it on the same side of the carriage as the primary carrier. Of course, this movement may incur further float-avoidant transfers.

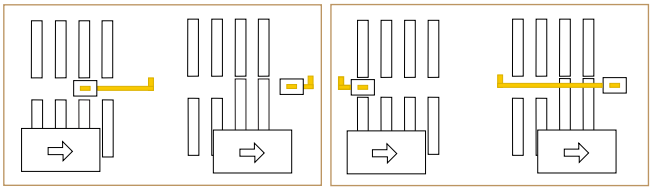


Fig. 16. From a top-down view, we depict two needle beds with four needles each, which move up into the gap to grab yarn and form loops. In each case, the carriage is moving left-to-right, as indicated by the arrow on it. There is one carrier for yellow yarn. Left: (Incorrect) When the carrier is anchored in front of the carriage in the carriage direction, the carrier stays in front of the carriage, and the yarn retracts rather than being laid over the needles. Right: (Correct) When the carrier is anchored behind the carriage, the needles can move into place before the carrier passes over them.

Algorithm 3 Modify carriage direction and course ordering or add instructions to set correct yarn path.

```
function SetYarnPath(colorCourse, rest, anchor, program)
    if anchor is closer to colorCourse start then
        carriageDir ← LeftToRight
    else
        carriageDir ← RightToLeft
    if anchor is between colorCourse start and end then
        if strategy is Split then
            flip carriageDir
            left, right ← split(colorCourse, anchor)
            if carriageDir is RightToLeft then
                colorCourse ← left
                rest += right
            else
                colorCourse ← right
                rest += left
        if strategy is Anchor then
            if carriageDir is LeftToRight then
                program += knitWTransfers(start, RightToLeft)
            else
                program += knitWTransfers(end, LeftToRight)
    if carriageDir is RightToLeft then
        reverse colorCourse
end function
```

8 EVALUATION

To demonstrate that our approach is successful, we created a broad variety of traditional and totally novel knitted illusions, showing that the design burden is low, the resulting illusions obviously exhibit the input images in each view, and actual fabrication is possible.

8.1 Automating Single-view Illusions

The Mona Lisa illusion in Figure 1 demonstrates our automation of single-view illusions: only one input image is required. Compared to the traditional approach requiring hundreds of hours of manual labor, our framework significantly expedites the process and enables non-experts to create intricate illusion knits.

Fig. 17. A single-view head-on illusion and its inputs. Viewed from the front, the dragon is visible, but when viewed at an angle, the larger circle illusion engulfs it, and only a circle is visible. A tile variation makes it possible to produce this other type of knit illusion.

8.2 Beyond Single-view Illusions

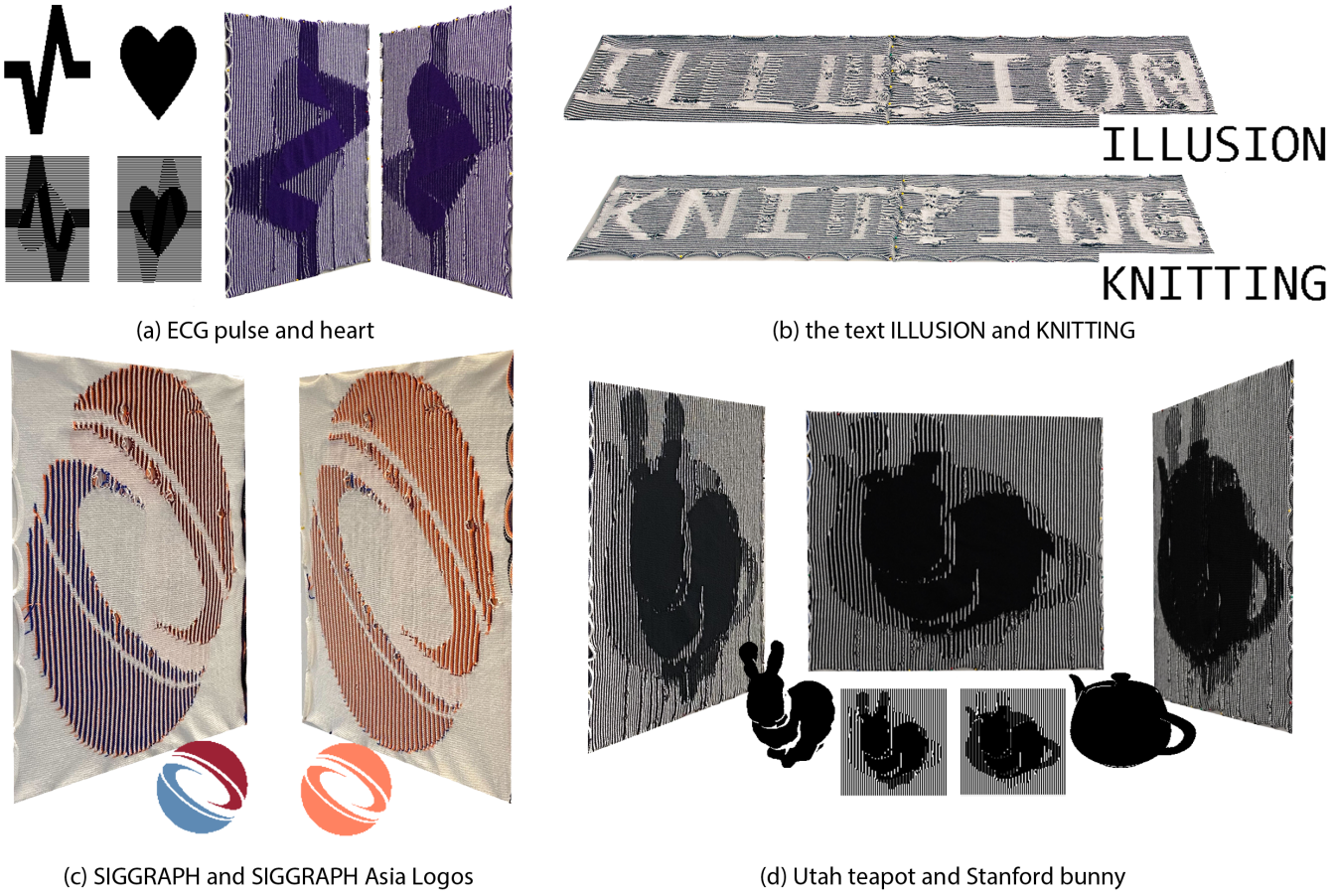
This paper expands the range of achievable illusions in knitting; in particular, we have detailed techniques for novel double-view illusions and provided the formal foundations for even more illusion types. We show examples generated with our system, all taking five to twenty minutes of design time to iterate on the input images, yarn choices, tile selection, and editing.

Figure 17 shows an example that can be viewed head-on but is occluded in a side view; this novel effect was enabled via tile editing in our design tool. We also produced the first known double-view illusions, where two two-color images are knitted using two yarn colors. In our examples, we choose the automated assignments that replace the background color with a striped pattern. This approach yields sharp results, as in the bunny and teapot (Figure 18 (d)) and the ECG pulse and heart (Figure 18 (a)). We demonstrate text legibility in a more challenging example, where one side reads "ILLUSION" and the other reads "KNITTING" (Figure 18 (b)).

We push the boundaries further with illusions using more than two yarn colors. In Figure 18 (c), we show the SIGGRAPH logo in red and blue on one side and an orange logo for SIGGRAPH Asia on the other side. This example uses four yarn colors, with a fixed background and changing logo colors. Finally, to even further push the limits of our techniques, we knit two complex Van Gogh paintings using a three-color quantization and three yarn colors, requiring a total of nine unique tiles that were automatically created and assigned (Figure 1). The different views are distinct, and the images are recognizable. This example demonstrates the potential of our technique.

8.3 Comparison with Other Design Methods

We also experimented with fully automatic methods for designing illusion patterns. We encoded constraints from subsection 4.2 as a MaxSAT formula and solved for the lowest-logical-error pattern. We



(a) ECG pulse and heart

(b) the text ILLUSION and KNITTING

(c) SIGGRAPH and SIGGRAPH Asia Logos

(d) Utah teapot and Stanford bunny

Fig. 18. Double-view examples, pictured with their inputs, and, in (a) and (d), the rendered prediction of their patterns. In each example, the image from the left and right viewing angle differs, as depicted. In (d), we also show the head-on view.

also used our differentiable renderer to perform gradient descent, using MSE for the loss. Figure 19 shows results of both.

These patterns are slow to knit: 12 hours for the MaxSAT-generated bunny and teapot vs 2 hours for our tiled solution, and an estimated 30 hours of knitting for the optimized Van Gogh vs 10 hours for our tiled solution (which is a conservative comparison, as our evaluation example is a larger piece). Additionally, we found the illusions from the automated techniques less obvious than our tiled approach. Other automated methods may improve clarity and lower fabrication times, but those can also be incorporated into our framework. In our experience, a designer-in-the-loop tool is highly effective for optimizing the ultimate objective of human-perceptible knit illusions.

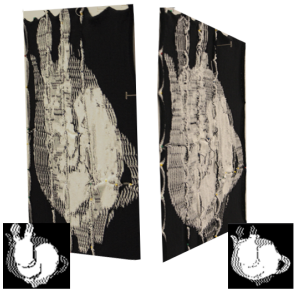
8.4 Fabrication

We implemented a knitting pattern to machine instruction pipeline, which lowers a logical design to a physical pattern, then compiles it to a knitout program. We use the KnitPaint program to generate machine code and knit the objects on a 7-gauge (i.e., 7 needles per inch) SWG091N2, which is a Shima Seiki WholeGarment knitting machine. Designs took between 50 minutes (ECG pulse and heart, 200 x 200 stitches) to ten hours (Van Gogh, 660 x 400 stitches). The fabrication time for single-view patterns is determined by the size and in double-view by pattern complexity. In addition to the illusion results, we show in Figure 20 what happens when each key feature of our fabrication algorithm (i.e., float avoidance, path planning, and catching) is not applied, demonstrating their importance. In Figure 21, we specifically highlight a full piece that has been knit without path planning.

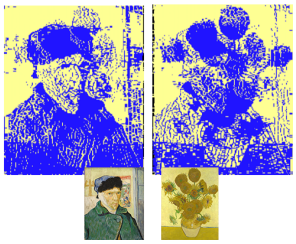
9 LIMITATIONS AND FUTURE WORK

Our work presents the first framework for users to explore the space of illusion knitting where more than one image can be hidden. We focus on how user-in-the-loop systems that systematically relax constraints with automated assistance can help users express their desired designs, and hope to inspire more work in this methodology.

In the future, the fabrication strategies we propose can be applied in situations beyond the mixed colorwork and texture needed for illusion knitting. Though the development of the yarn anchoring strategy was motivated by our inability to use the back bed at will



(a) The result of using our MaxSAT formulation to solve for a pattern that satisfies the most constraints. Notice how the areas where the two images differ still have a lot of visual interference. This pattern took 10 hours to knit.



(b) The result (large images) of running optimization using our differentiable renderer and MSE loss against the target inputs (small pictures) to get a double-view illusion design. Although the predicted result looks reasonable, this design would have taken 30 hours to knit.

Fig. 19. Examples of knit illusions produced by other design tools. Regardless of the quality of these automated approaches, a human-in-the-loop tool is still necessary so designers have recourse to correct or modify parts of the design.

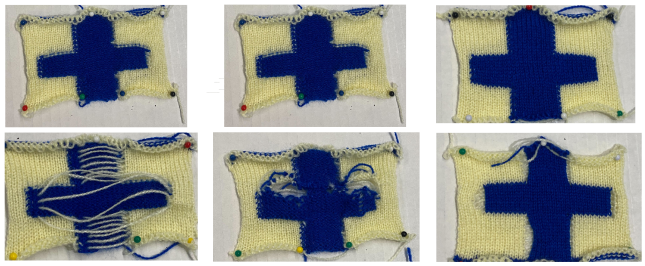


Fig. 20. Knitting with (top row) and without (bottom row) each contribution. Without float avoidance (left), floats appear on the front of the work. Without path planning (middle), the piece systematically dropped stitches. Without catching (right), the two pieces knit with different colors are disconnected at the vertical edges, leading to the blue sides of the plus sign to curl inwards.

Fig. 21. An attempted illusion knit without path planning. Note that the failure manifests not as mere visual artifacts on the knit object, but in the fact that the knit is structurally not a surface at all on the right side, and nothing is viewable.

in illusion knitting, this advancement could enable simpler intarsia scheduling for stockinette garments since plating is simple to reason about. Our yarn path planning strategy might also be used for intarsia scheduling as well as more broadly when considering how to schedule complex objects.

Our contributions also open several other avenues for future work. First, we considered the space of illusions achieved only through standard knits and purls on a knitting machine. More complex fabrication techniques expand the possibilities. For example, knitting a color change into a purl bump can partially achieve different colors on each side of the bump. Alternatively, expert machine programmers might attempt to enhance the height difference between bump and flat microgeometry by adding extra yarn to bumps via additional tucks, or by mimicking e-wraps, or another similar strategy. New design tools would be essential to understanding this more complex design space and to make design decisions that trade off different aspects, e.g., physical limitations of yarns, fabrication methods, human perception, and visual complexity.

Second, there may be other techniques for machine knitting that we could have used to further improve fabrication. For example, we did not consider using sliders independently from their needles, which can hold loops temporarily and could catch yarn for intarsia and for anchoring. This solution would call for more complex scheduling algorithms, perhaps when the aesthetics of the knitted piece is a larger consideration.

Our knits tend to have small areas of dropped stitches. We believe this is due to the extra transfers incurred, each adding a chance that loops will fall off the needle. Investigation into reducing transfers would be valuable for reducing the number of dropped stitches.

It is clear that illusions present interesting challenges in computational knitting and provide a rich case study for novel methods in pattern design, program scheduling, and interaction. Our efforts serve as a framework for working with more general illusion knitting, and we intend to continue expanding its functionality.

10 CONCLUSION

We developed a theoretical understanding of illusion knitting by leveraging the key insight that both the microgeometry that underpins illusions and the principles behind illusions can be expressed as constraints. Combined with practices from artists in the craft, we built a human-in-the-loop design framework, focused around tiles, for creating illusion knitting patterns that reduce illusion design to familiar image editing tasks. To fabricate these complex designs combining colorwork and texture, we introduced novel machine knitting scheduling techniques. We further developed completely new kinds of illusion designs and so establish a foundation for the community to further explore illusion knitting.

ACKNOWLEDGMENTS

We would like to thank Raymond Guo, Kaiyang Wu, and Jimmy Cheong for their contributions to the MaxSAT formulation. We also thank Kevin Mu for helping with rendering several of the figures. Finally, we thank the anonymous reviewers for their thoughtful and thorough feedback that has improved the paper immensely. This work was funded by NSF 2017927 and NSF 2319181.

REFERENCES

Lubna Abu Rmaileh and Alan Brunton. 2023. Meso-Facets for Goniochromatic 3D Printing. *ACM Trans. Graph.* 42, 4, Article 66 (jul 2023), 12 pages. https://doi.org/10.1145/3592137

Lea Albaugh, Scott Hudson, and Lining Yao. 2019. Digital Fabrication of Soft Actuated Objects by Machine Knitting. In *Extended Abstracts of the 2019 CHI Conference on Human Factors in Computing Systems*. ACM, Glasgow Scotland Uk, 1–4. https://doi.org/10.1145/3290607.3313270

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

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*. ACM, Yokohama Japan, 1–12. https://doi.org/10.1145/3411764.3445564

Marc Alexa and Wojciech Matusik. 2011. Images from Self-Occlusion. In *Proceedings of the International Symposium on Computational Aesthetics in Graphics, Visualization, and Imaging* (Vancouver, British Columbia, Canada) *(CAe '11)*. Association for Computing Machinery, New York, NY, USA, 17–24. https://doi.org/10.1145/2030441.2030445

Marc Alexa and Wojciech Matusik. 2012. Irregular pit placement for dithering images by self-occlusion. *Computers & Graphics* 36, 6 (2012), 635–641.

Amit Bermano, Ilya Baran, Marc Alexa, and Wojciech Matusk. 2012. ShadowPix: Multiple Images from Self Shadowing. *Computer Graphics Forum* 31, 2pt3 (May 2012), 593–602. https://doi.org/10.1111/j.1467-8659.2012.03038.x

Teresa Harmon. 2011. *A Guide to Illusion Knitting: It's not magic, it's just fun!* CreateSpace Independent Publishing Platform, US.

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* (<conf-loc>, <city>San Francisco</city>, <state>CA</state>, <country>USA</country>, </conf-loc>) *(UIST '23)*. Association for Computing Machinery, New York, NY, USA, Article 21, 21 pages. https://doi.org/10.1145/3586183.3606789

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*. ACM, Virtual Event USA, 1234–1247. https://doi.org/10.1145/3379337.3415590

Vivian Hoxbro. 2004. *Shadow Knitting*. Interweave, Loveland, Colo.

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

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

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/10.1145/3476576.3476614

Jenny Lin and James McCann. 2021. An Artin Braid Group Representation of Knitting Machine State with Applications to Validation and Optimization of Fabrication Plans. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, Xi'an, China, 1147–1153. https://doi.org/10.1109/ICRA48506.2021.9562113

Jenny Lin, Vidya Narayanan, Yuka Ikarashi, Jonathan Ragan-Kelley, Gilbert Bernstein, and James McCann. 2023. Semantics and Scheduling for Machine Knitting Compilers. *ACM Trans. Graph.* 42, 4 (Aug. 2023), 17. https://doi.org/10.1145/3592449 In Press.

Jenny Lin, Vidya Narayanan, and James McCann. 2018. Efficient transfer planning for flat knitting. In *Proceedings of the 2nd ACM Symposium on Computational Fabrication*. ACM, Cambridge Massachusetts, 1–7. https://doi.org/10.1145/3213512.3213515

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 (Aug. 2021), 1–16. https://doi.org/10.1145/3450626.3459868

Yiyue Luo, Kui Wu, TomÃ¡s 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*. ACM, Yokohama Japan, 1–12. https://doi.org/10.1145/3411764.3445780

Yiyue Luo, Kui Wu, Andrew Spielberg, Michael Foshey, Daniela Rus, TomÃ¡s 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

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), 1–11. https://doi.org/10.1145/2897824.2925940

Niloy J. Mitra and Mark Pauly (Eds.). 2009. Shadow art. *ACM Transactions on Graphics* 28, 5 (2009), 1–7. https://doi.org/10.1145/1661412.1618502

Rahul Mitra, Liane Makatura, Emily Whiting, and Edward Chien. 2023. Helix-Free Stripes for Knit Graph Design. In *ACM SIGGRAPH 2023 Conference Proceedings*. 1–9.

Georges Nader, Yu Han Quek, Pei Zhi Chia, Oliver Weeger, and Sai-Kit Yeung. 2021. KnitKit: a flexible system for machine knitting of customizable textiles. *ACM Transactions on Graphics* 40, 4 (Aug. 2021), 1–16. https://doi.org/10.1145/3450626.3459790

Sadako Nakamura, Ryo Tsugawa, Keiko Kitao, and et al. 1982. かくし絵ニット:やさしい棒針編のマジック *[Hidden Picture Knit : The Magic of Gentle Needle Knitting]*. Nihon Vogue, Tokyo.

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

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

orochi. 2006. かくし絵ニット・パターン作成ツールVersion 1.1 [Hidden Picture Knit・Pattern Making Tool]. https://orochiknit.com/archive/kt001.html

Marios Papas, Wojciech Jarosz, Wenzel Jakob, Szymon Rusinkiewicz, Wojciech Matusik, and Tim Weyrich. 2011. Goal-based Caustics. *Computer Graphics Forum* 30, 2 (2011), 503–511. https://doi.org/10.1111/j.1467-8659.2011.01876.x arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1111/j.1467-8659.2011.01876.x

Hao Peng, Lin Lu, Lin Liu, Andrei Sharf, and Baoquan Chen. 2019. Fabricating QR codes on 3D objects using self-shadows. *Computer-Aided Design* 114 (2019), 91–100.

Thiago Pereira, Carolina LA Paes Leme, Steve Marschner, and Szymon Rusinkiewicz. 2017. Printing anisotropic appearance with magnetic flakes. *ACM Transactions on Graphics (TOG)* 36, 4 (2017), 1–10.

Maxine Perroni-Scharf and Szymon Rusinkiewicz. 2023. Constructing Printable Surfaces with View-Dependent Appearance. In *Special Interest Group on Computer Graphics and Interactive Techniques Conference Conference Proceedings (SIGGRAPH '23 Conference Proceedings)*. ACM, New York, NY, USA, 10. In Press.

Mariana Popescu, Matthias Rippmann, Tom Van Mele, and Philippe Block. 2018. Automated Generation of Knit Patterns for Non-developable Surfaces. In *Humanizing Digital Reality: Design Modelling Symposium Paris 2017*, Klaas De Rycke, Christoph Gengnagel, Olivier Baverel, Jane Burry, Caitlin Mueller, Minh Man Nguyen, Philippe Rahm, and Mette Ramsgaard Thomsen (Eds.). Springer, Singapore, 271–284. https://doi.org/10.1007/978-981-10-6611-5_24

Christian Regg, Szymon Rusinkiewicz, Wojciech Matusik, and Markus Gross. 2010. Computational highlight holography. *ACM Transactions on Graphics (TOG)* 29, 6 (2010), 1–12.

Kaisei Sakurai, Yoshinori Dobashi, Kei Iwasaki, and Tomoyuki Nishita. 2018. Fabricating reflectors for displaying multiple images. *ACM Transactions on Graphics (TOG)* 37, 4 (2018), 1–10.

Yuliy Schwartzburg, Romain Testuz, Andrea Tagliasacchi, and Mark Pauly. 2014. High-contrast computational caustic design. *ACM Transactions on Graphics* 33, 4 (July 2014), 74:1–74:11. https://doi.org/10.1145/2601097.2601200

Pengfei Shen, Ruizeng Li, Beibei Wang, and Ligang Liu. 2023. Scratch-based Reflection Art via Differentiable Rendering. *ACM Transactions on Graphics (Proceedings of SIGGRAPH 2023)* 42, 4 (2023), 1–12.

Xavier Snelgrove, Thiago Pereira, Wojciech Matusik, and Marc Alexa. 2013. Parallax Walls: Light fields from occlusion on height fields. *Computers & graphics* 37, 8 (2013), 974–982.

Debbie Stoller. 2004. *Stitch 'n Bitch: The Knitter's Handbook*. Workman Publishing Company, New York.

Tim Weyrich, Pieter Peers, Wojciech Matusik, and Szymon Rusinkiewicz. 2009. Fabricating microgeometry for custom surface reflectance. *ACM Transactions on Graphics (TOG)* 28, 3 (2009), 1–6.

WoollyThoughts. 2023. Information. http://www.illusionknitting.woollythoughts.com/information.html

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

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

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

Jiani Zeng, Honghao Deng, Yunyi Zhu, Michael Wessely, Axel Kilian, and Stefanie Mueller. 2021. Lenticular Objects: 3D Printed Objects with Lenticular Lens Surfaces That Can Change Their Appearance Depending on the Viewpoint. In *The 34th Annual ACM Symposium on User Interface Software and Technology* (Virtual Event, USA) *(UIST '21)*. Association for Computing Machinery, New York, NY, USA, 1184–1196. https://doi.org/10.1145/3472749.3474815