FabObscura: Computational Design and Fabrication for Interactive Barrier-Grid Animations

Ticha Sethapakdi
MIT CSAIL
Cambridge, MA, USA

Maxine Perroni-Scharf
MIT CSAIL
Cambridge, MA, USA

Mingming Li
Zhejiang University
Hangzhou, China

Jiaji Li
MIT CSAIL
Cambridge, MA, USA

Justin Solomon
MIT CSAIL
Cambridge, MA, USA

Arvind Satyanarayan
MIT CSAIL
Cambridge, MA, USA

Stefanie Mueller
MIT CSAIL
Cambridge, MA, USA

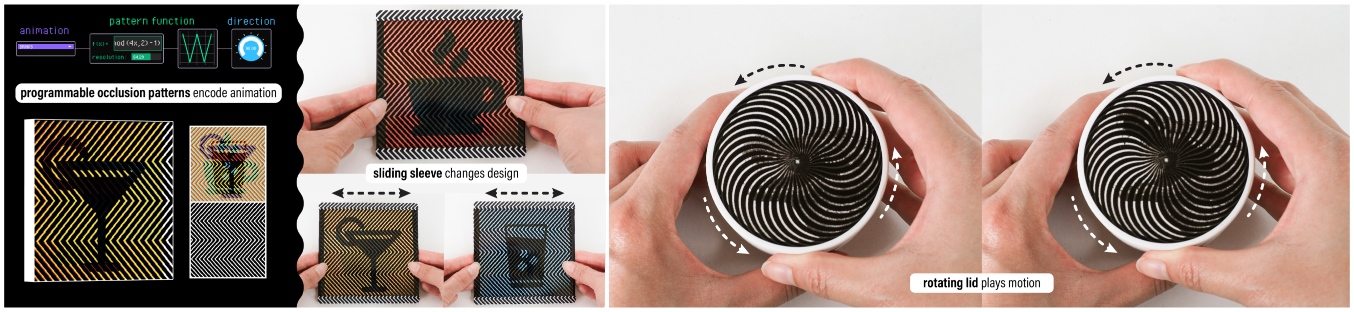
animation
DRINKS
pattern function
f(x)= nod(4x,2)-1)
resolution: 64.29
direction
90.00
programmable occlusion patterns encode animation
sliding sleeve changes design
rotating lid plays motion

Figure 1: *FabObscura* is a system for creating visually dynamic physical media based on the classic barrier-grid animation technique. We introduce a novel parameterization and computational design tool for systematically designing new barrier-grid animations without domain expertise. Our abstraction is expressive enough to support animations that respond to diverse user interactions, such as translations, rotations, and changes in viewpoint.

Abstract

We present *FabObscura*: a system for creating interactive barrier-grid animations, a classic technique that uses occlusion patterns to create the illusion of motion. Whereas traditional barrier-grid animations are constrained to simple linear occlusion patterns, *FabObscura* introduces a parameterization that represents patterns as mathematical functions. Our parameterization offers two key advantages over existing barrier-grid animation design methods: first, it has a high expressive ceiling by enabling the systematic design of novel patterns; second, it is versatile enough to represent all established forms of barrier-grid animations.

Using this parameterization, our computational design tool enables an end-to-end workflow for authoring, visualizing, and fabricating these animations without domain expertise. Our applications demonstrate how *FabObscura* can be used to create animations that respond to a range of user interactions, such as translations, rotations, and changes in viewpoint. By formalizing barrier-grid animation as a computational design material, *FabObscura* extends its expressiveness as an interactive medium.

ACM Reference Format:
Ticha Sethapakdi, Maxine Perroni-Scharf, Mingming Li, Jiaji Li, Justin Solomon, Arvind Satyanarayan, and Stefanie Mueller. 2025. *FabObscura*: Computational Design and Fabrication for Interactive Barrier-Grid Animations. In *The 38th Annual ACM Symposium on User Interface Software and Technology (UIST '25), September 28-October 1, 2025, Busan, Republic of Korea.* ACM, New York, NY, USA, 16 pages. https://doi.org/10.1145/3746059.3747629

1 Introduction

Physical occlusions, or barriers, are integral components of our daily environments, manifesting as window blinds, curtains, fencing, packaging, and architectural facades. Historically, barriers have also played a role in animation through aptly-named *barrier-grid animations*, which employ occlusion patterns to encode animation frames and produce the illusion of movement [20, 28, 32]. Conceived in the 19th century, barrier-grid animations offer a means of presenting dynamic content without relying on electronics or advanced materials. In the earliest forms of barrier-grid animations, users "played" the animation by sliding a striped transparent overlay across an interlaced image that encodes the animation frames [21, 38].

Over the years, barrier-grid animations have been adapted to accommodate a range of materials, form factors, and interactions. Visual artists and designers have leveraged the technique to create a variety of visually dynamic interactive artifacts, from books with illustrations that animate as you turn the page [30], to analog clocks with dynamic faces that shift with each tick [37], to 10-foot-tall murals that morph as you walk past them [6]. These applications demonstrate how barrier-grid animations can be designed to respond to different interaction modalities: translations, rotations, and changes in viewpoint.

Although barrier-grid animations offer a promising approach for turning physical occlusions into dynamic design materials, they currently exhibit a very limited visual vocabulary. Designs predominantly rely on linear occlusion patterns (i.e., straight vertical, horizontal, or radial lines), which constrains the range of achievable visual effects and motion qualities. While some experienced optical artists like Rufus Butler Seder [11] and Gianni Sarcone [26] have developed a handful of novel barrier-grid designs as part of their creative practice, their methods remain as tacit knowledge that is neither formalized nor easily reproducible without significant domain expertise.

The absence of systematic design methods and prototyping tools fundamentally constrains what creators can achieve with barrier-grid animations. Designers are caught between either using specialized tools that restrict creative exploration, or general-purpose software operate at incorrect levels of abstraction. Several online tools exist for creating barrier-grid animations [4, 17, 24]. However, each is tailored to a single type of animation and prevents users from designing novel patterns or exploring different animation styles. Alternatively, users may construct animations manually with imaging software such as Adobe Photoshop—however, such general-purpose tools operate at the pixel level rather than shape the higher-level patterns and behaviors that govern barrier-grid animations. This discrepancy makes iterative exploration challenging and inhibits creative experimentation with the medium.

Current limitations in barrier-grid animation creation processes pose substantial barriers to entry for both novices exploring the medium and experienced designers seeking to advance it as a form of visual expression. This presents an opportunity for computational tools that can both lower the *threshold* for newcomers and raise the *expressive ceiling* [19] for seasoned artists, enabling barrier-grid animations to be more broadly adopted as a design material.

This paper introduces *FabObscura*, a design and fabrication system for interactive barrier-grid animations. By identifying and formalizing the canonical properties of barrier-grid animations, we derive a novel parameterization that significantly extends the technique's expressive capabilities. Our parameterization enables a computational design and fabrication workflow for designers to experiment with and create new barrier-grid animation forms without domain expertise. In contrast to existing workflows, our technique supports a high expressive ceiling (i.e., it provides a means of systematically creating new barrier designs) and *low-viscosity* [8] (i.e., it allows users to explore designs under different representations with low friction). To demonstrate how *FabObscura* enhances the expressivity of physical objects, we present six fabricated barrier-grid animations integrated into various form factors.

Our contributions are:

(1) A *novel parameterization* of barrier-grid animations that enables the creation of diverse occlusion patterns;
(2) A *design tool* for authoring, visualizing, and fabricating different types of barrier-grid animations;
(3) A *technical evaluation* that quantifies the trade-offs between different pattern parameters and provides guidelines for optimal visual quality;
(4) Six fabricated *applications* that showcase how our technique can support expressive interactions and dynamic visual communication across a range of form factors.

2 Related Work

FabObscura builds upon research on fabrication systems for physical artifacts whose visual appearance changes in response to external stimuli.

2.1 Fabricating Visually Dynamic Objects

Fabricating physical objects with dynamic appearances, such as view-dependent visuals or interactive color-changing effects, has been a central focus in HCI and computational fabrication research. This line of work explores how to imbue physical artifacts with changing visual properties that respond to user interaction or environmental stimuli [9].

Researchers have devised various techniques for making color-changing displays from materials that react to changes in environmental conditions. For instance, thermochromic inks, which change color with temperature, have been used to produce temperature-responsive motion effects [48] and multicolored shifting images [33]. Similarly, researchers have applied photochromic inks, which change color when exposed to ultraviolet light, to create surfaces and objects with dynamically reprogrammable appearances [12, 51]. In *Polagons* [31], birefringent materials produce mosaic displays that change appearance based on incident polarized light.

Other techniques focus on precisely modifying a material's surface microstructure to produce view-dependent effects. For example, lenticular lenses are microstructured optical elements consisting of an array of magnifying lenses that direct light in specific ways based on viewing angle. They have been used to create three-dimensional objects with viewpoint-dependent appearances [50], and even dynamic gastronomic experiences in which food visually transforms [46]. Pjanic and Hersch [23] create metallic, light-dependent visual effects by printing anisotropic halftones (i.e., small colored ink lines) onto metal surfaces. These halftones produce different colors depending on the orientation of the surface relative to the light source, resulting in distinct visual outputs at 90-degree rotations. Shen et al. [34] employ a differential rendering-based optimizer to create scratch patterns on metallic surfaces, causing them to reflect different light fields based on the direction of the incident light. This enables viewers to perceive different images depending on their viewing position. Levin et al. [14] use a multi-step process involving etching and photolithography to create custom Bidirectional Reflectance Distribution Functions (BRDFs) that modify a surface's appearance based on the viewing angle.

Finally, researchers have developed a range of support tools for making artifacts that animate through mechanical motion. For example, researchers developed computational design and fabrication systems that allow non-expert users to create complex kinematic artifacts [5, 15, 16]. Miyashita et al. [18] and Kushner et al. [13] applied the classic Zoetrope animation technique to create mechanically actuated displays that reveal three-dimensional frame-by-frame animations. Complementary to these approaches, *FabObscura* draws inspiration from barrier-grid animation to produce a range of visual effects—including color-change and motion—that are triggered by simple mechanical movements like sliding and rotation.

2.2 Occlusion-Based Visual Effects

Our work leverages the natural properties of occlusion to create the illusion of motion. Self-occlusion, where certain parts of an object block others depending on the viewing angle, is a common technique for creating visual effects by manipulating geometry.

Several previous studies have explored self-occlusion as part of various physical displays. Sakurai et al. [25] use fixed-height fields to create self-occlusion effects over colorful subcells, while Abu Rmaileh and Brunton [1] generate color variations across 3D surfaces using meso-faceted heightfields. Expanding on these approaches, Perroni-Scharf and Rusinkiewicz [22] introduce a gradient-descent optimization that fine-tunes height and color within a heightfield to achieve specific view-dependent effects. In all these methods, subtle height variations are used to manipulate occlusion, light and shadow, resulting in visually dynamic surfaces.

In addition, several studies have explored how self-occlusion can enhance lighting-dependent effects. *Parallax Walls* [36] create light fields from small occluding walls, with the display's color shifting depending on the direction of the incident light. Similarly, *ShadowPix* [3] uses an optimization algorithm to adjust wall heights, casting shadows that change with varying light angles. Yamamoto and Sugiura [44] apply this principle to carpets, transforming them into switchable multi-image displays by manipulating fiber direction to create different shades based on the position of the light source or the viewer's perspective. In a related approach, Wetzstein et al. [43] explore the use of layered transparent light modulators to generate light fields, making 2D displays appear 3D.

2.3 Applications of Dynamic Physical Media

Dynamic physical media are gaining popularity in HCI due to their ability to offer novel interactions to real world objects. Recent advances in display fabrication provide valuable insights into integrating interactivity and personalization [10, 42, 45], enabling dynamic media to be used for a range of practical applications.

In educational settings, dynamic media have been used to create interactive and hands-on learning experiences. Reflection holographic displays have been used for 3D anatomical diagrams for medical students, improving spatial reasoning [39]. Research indicates that dynamic, multi-dimensional visual aids engage students more effectively and improve learning outcomes compared to traditional 2D materials [47].

Dynamic media can also provide valuable real-time feedback and visual cues. Ronchi and Moiré patterns, for instance, are commonly employed in surface defectometry and geometric analysis due to their ability to encode information in parallel striations that shift with viewing angles [2]. These patterns have been applied to 3D objects to extract geometric data for critical robotics tasks such as pose estimation and object tracking. *Moiré Widgets* [49] demonstrated how to use Moiré patterns to create battery-less tangible interfaces with interactive capabilities.

3 Background: Barrier-Grid Animations

FabObscura is based on the principles underlying barrier-grid animations, which are so-called for their use of occlusions, or "barriers", to selectively hide or reveal portions of an animated sequence. In this section, we explain the working principles of barrier-grid animations, survey their known variants, and identify canonical properties that unify all variants.

3.1 Basic Construction

Fundamentally, barrier-grid animations have two components: an *interlaced image* and a *barrier pattern*. A barrier represents a striped pattern composed of opaque and transparent regions. An interlaced image is a composite image that encodes an animation sequence by slicing multiple individual frames into equally sized strips and stacking them in an alternating sequence (Fig. 2).

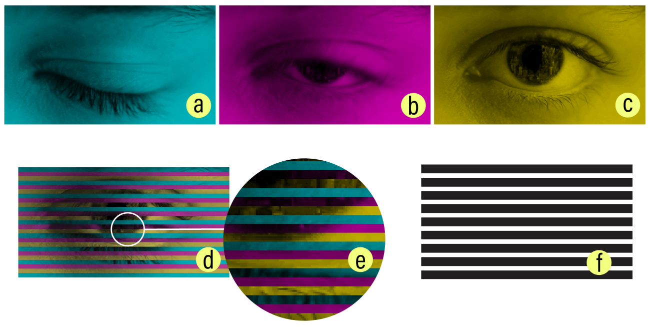

Figure 2: The interlacing process for a (a-c) three frame animation depicting an opening eye. The (d) interlaced image is created by (e) slicing and combining strips from each frame in an alternating sequence. The corresponding (f) barrier pattern consists of opaque stripes that block two frames while its transparent regions reveal the third frame.

For a barrier-grid animation to work correctly, the interlaced frame strips and barrier must be properly aligned so that, when the barrier is placed on top, it masks all frames except for the one currently being viewed. This imposes two **design constraints**:

(1) If the *interlacing direction* determines the orientation of the interlaced image strips, the barrier stripes must follow the same direction to ensure proper masking;
(2) In an animation with n frames, if each interlaced frame strip has width w, the barrier must alternate between transparent stripes of width w and opaque stripes of width $(n-1)w$ to ensure that only one frame is visible at a time.

Violating either of these constraints results in an undesired effect called *ghosting*. This occurs when portions of frames other than the intended frame of interest are visible at a given view, resulting in faint ghost-like interference patterns becoming visible on top of the main image shown.

Next, we present a typology of barrier-grid animations, categorizing them based on their interaction mechanisms and the distinct visual effects that they produce.

3.2 Sliding Animations

The oldest and most widely-used variant of barrier-grid animations relies on **sliding interactions**. Typical sliding animations make use of horizontally- or vertically-oriented barriers. The barrier's orientation determines the interlacing direction as well as the primary axis of motion: when a barrier is placed on top of an interlaced image, sliding it along the interlacing direction causes the transparent regions to sequentially reveal different portions of the underlying animation frames (Fig. 3). Continuing to slide the barrier makes the animation loop back to the first frame after revealing the last.

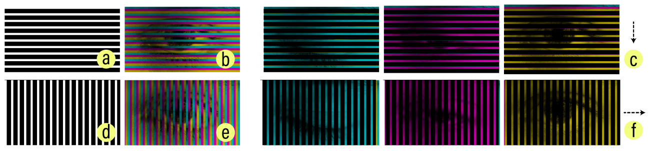

Figure 3: Sliding the (a) horizontal barrier down the (b) horizontally-interlaced image (c) sequentially reveals different frames. Similarly, sliding a (d) vertical barrier across a (e) vertically-interlaced image interlaced sequentially reveals (f) the underlying animation.

3.3 View-Dependent Animations

While traditional barrier-grid animations require users to physically slide its constituent components, artists devised a technique that supports **view-dependent interactions** [6, 29]. View-dependent animations maintain a fixed separation between the interlaced image and barrier layers within a connected structure. As users change their viewing angle—either by repositioning themselves or tilting the structure—the resulting parallax effect produces the same sequential animation as sliding animations (Fig. 4).

Although view-dependent animations have the advantage of providing a "hands-free" interaction experience, they also introduce new alignment challenges. Increasing the distance between the interlaced image and barrier increases the amount of parallax, meaning that a smaller change in the viewing angle is required to advance to the next frame. However, as the distance between the interlaced image and barrier layers increases, they gradually become more misaligned due to perspective effects (e.g., the barrier may appear much larger than the interlaced image). Thus to mitigate ghosting, designers must manually adjust the size of the interlaced image to maintain proper alignment between layers.

3.4 Rotational Animations

As previously discussed, the barrier orientation and corresponding interlacing direction determine the animation's direction of

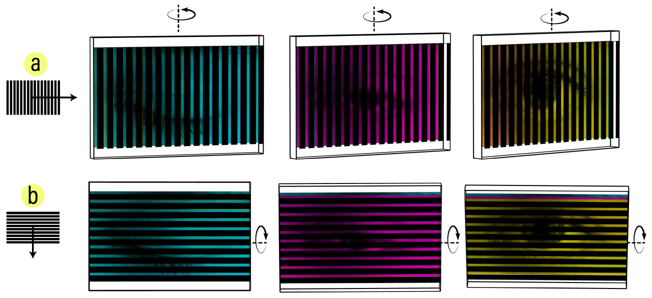

Figure 4: Viewing a barrier-grid animation from different angles produces the same effect as sliding the barrier. (a) Moving left (or tilting counterclockwise) simulates sliding a vertical barrier right. (b) Viewing from above (or tilting downward) simulates sliding a horizontal barrier down.

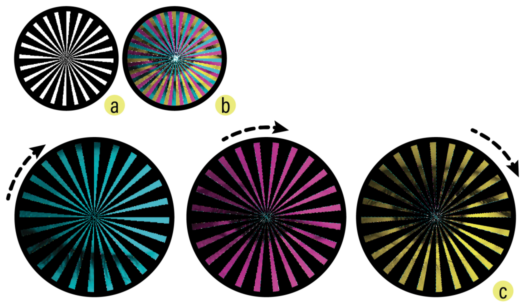

Figure 5: Rotational barrier-grid animations feature (a) a radial barrier pattern and (b) radially interlaced image with frames arranged in alternating wedges. Rotating the barrier pattern clockwise sequentially reveals (c) underlying frames.

motion. Whereas sliding and view-dependent animations are linearly interlaced along the x or y axes, we can also perform radial interlacing along the polar axis. This yields a construction that facilitates **rotational interactions** [27]. Under this scheme, the barrier pattern and interlaced image have radial symmetry and comprise wedge-shaped segments that extend from the center (Fig. 5). Rotating a barrier clockwise around the center advances the underlying animation.

3.5 Canonical Properties

From our analysis of barrier-grid animation types, we see that the barrier pattern significantly influences the aesthetics of the animation and achievable visual effects. However, existing patterns are largely limited to straight horizontal, vertical, or radial lines. Generalizing these patterns into more diverse forms is therefore key to extending expressivity.

The first step to generalizing barrier-grid animations is defining a set of canonical properties that any barrier pattern variant must satisfy. From closely studying the variants and their design constraints, we identified three fundamental properties:

(1) **Seriality**: The pattern sequentially reveals corresponding strips of different frames as it moves along a fixed direction;

(2) **Periodicity**: The pattern repeats at regular intervals and aligns precisely with the interlacing period, returning to the first frame after the last;
(3) **Discreteness**: The transparency distribution of the pattern ensures that at most one frame is fully visible at each time step.

Together, these properties provide a conceptual framework for the space of valid barrier patterns that applies to all types of barrier-grid animations.

4 Design Space

In this section, we establish the design space that guides our computational approach to barrier-grid animation.

4.1 Parameterizing Patterns

Guided by our canonical properties from Section 3.5, we can build a parameterization that encompasses the space of valid barrier patterns. To satisfy *seriality*, the barrier pattern must be aligned to the interlacing path, i.e., the shape of each barrier segment must match the shape of each interlaced strip. To satisfy *periodicity*, each barrier segment must be duplicated multiple times to form a repeating pattern. Thus, we can conceptualize any barrier pattern as a sequence of **pattern units** that are repeatedly stacked along the axis corresponding to a given direction of motion. We preserve *discreteness* by making the stack alternate between opaque and transparent units.

Defining the Shape of a Pattern Unit. Consider a linear (i.e., sliding or view-dependent) animation where the direction of motion is vertical. This is equivalent to movement along the y-axis in Cartesian space. For an interlacing to be valid, observe that each interlaced strip must be uniquely identifiable at any x position—meaning a vertical line passing through the image at any x coordinate must intersect each strip exactly once. This property is analogous to the "vertical line test" in mathematics, which determines whether a graph represents a function: for each unique input (x-value), there must be exactly one output (y-value).

Conveniently, this mapping between valid interlaced strips and functions allows us to parameterize our pattern unit with *any* mathematical function f that describes its path as it moves along the x-axis (Fig. 6). For example, the function $f(x) = C$ (where C is a constant) creates straight horizontal lines, while $f(x) = \sin(x)$ produces a wavy pattern.

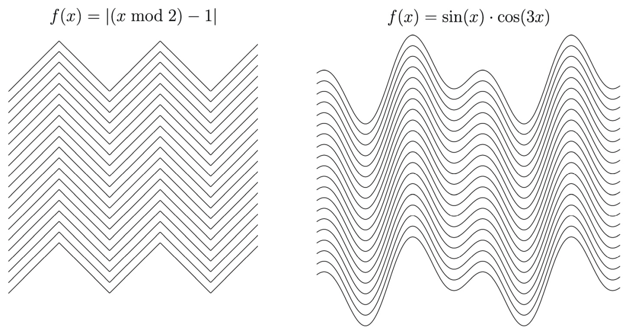

Figure 6: Examples of valid patterns for linear animations.

The same one-to-one mapping principle holds for rotational animations, though now expressed in polar coordinates rather than Cartesian space. In this framework, the path of each radial line in the pattern is governed by a function f that defines the local inclination (i.e., the turning angle) as a function of r, the radial distance measured outward from the center of rotation. As r increases, $f(r)$ controls how much the line bends at each step (Fig. 7). For example, the function $f(r) = 0$ creates straight radial lines, while $f(r) = C$ (where C is a nonzero constant) produces arcs that spiral out from the center. Importantly, any circular path centered at the origin intersects each radial line exactly once.

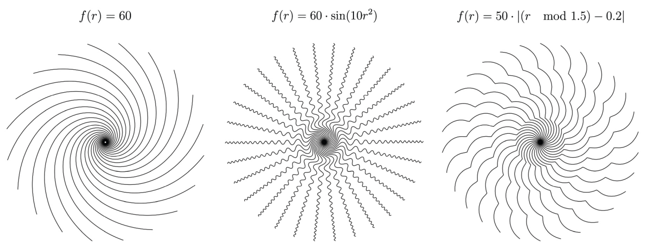
$f(r) = 60$
$f(r) = 60 \cdot \sin(10r^2)$
$f(r) = 50 \cdot |(r \mod 1.5) - 0.2|$

Figure 7: Examples of valid patterns for radial animations.

Choosing the Direction of Motion. Since pattern units in rotational animations are stacked along the radial axis, their direction of motion always follows a circular path. Linear animations, on the other hand, allow us to define the direction of motion by rotating pattern units before stacking them (Fig. 8). For example, rotating a constant function $f(x) = C$ by 90° changes the direction of motion from vertical to horizontal by transforming a horizontal striped barrier into a vertical striped barrier, while rotating by 45° makes the direction of motion diagonal.

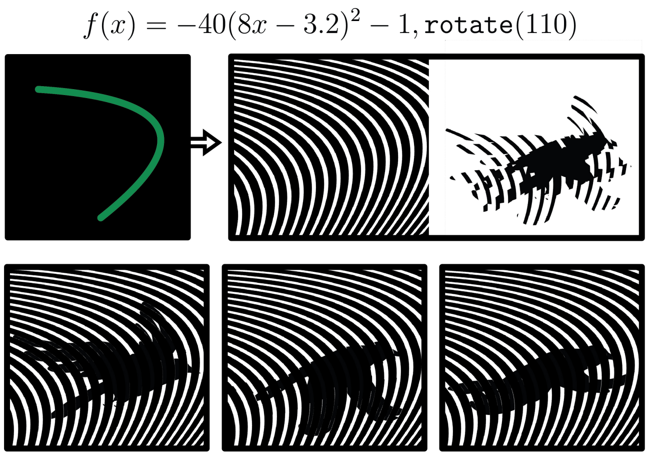

Figure 8: Patterns can be rotated to specify a direction of motion and/or add visual interest to certain parts of the animation. Here, we rotate a quadratic function by 110° to emphasize the direction that the bird is flying in.

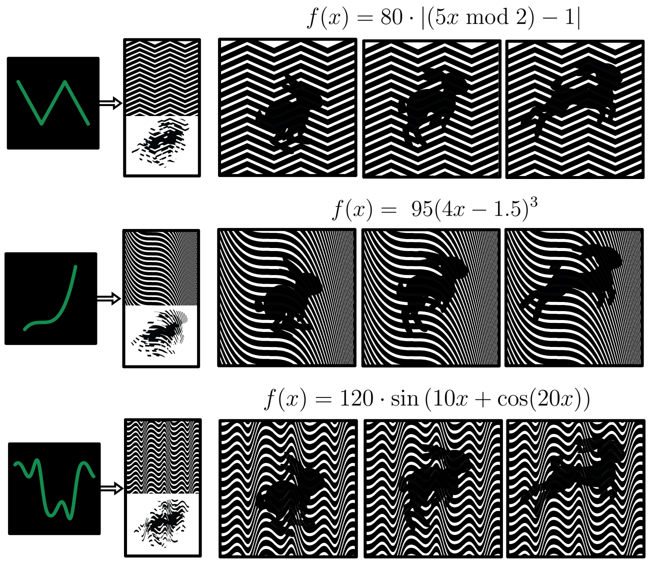
$f(x) = 80 \cdot |(5x \bmod 2) - 1|$
$f(x) = 95(4x - 1.5)^3$
$f(x) = 120 \cdot \sin(10x + \cos(20x))$

Figure 9: An animation of a jumping rabbit interlaced with three different barrier unit functions at the same resolution. Each pattern animates along the y-axis.

Design Parameters. In summary, we use the following parameterization for our barrier patterns:

(1) **Unit Function** f: The mathematical function that governs the shape of each pattern unit.
(2) **Thickness** t: A value used to calculate the thickness of the opaque and transparent units. For an animation with n frames, the pattern alternates between transparent units of thickness t and opaque units of thickness $(n-1)t$.
(3) **Direction** θ: The angle that determines the linear animation's direction of motion.

Our parameterization provides two key advantages over conventional methods for barrier-grid animation generation. First, it has a *high expressive ceiling* as it provides a framework for systematically generating entirely new forms with predictable behavior. Figure 9 shows several novel patterns alongside their corresponding functions, illustrating how different mathematical formulations extend the visual vocabulary of barrier-grid animations while preserving their core properties.

Second, as our parameterization is derived from core barrier grid animation principles, it generalizes across all barrier-grid animation variants and provides a means to switch between different representations with *low viscosity*. Figure 10 shows how the same unit function produces a valid animation for all three variants, which demonstrates the coherence of our parameterization across different interaction modes.

4.2 Nesting Animations

Interlaced images can be used to encode not only sequences of animation frames, but sequences of entire barrier-grid animations. This enables *nested* animations, where one barrier-grid animation is embedded within another (Fig. 11). In such compositions, multiple barrier patterns operate together, each selectively revealing different layers of a complex interlaced image. A primary barrier may determine which animation is visible, while secondary barriers control the progression of frames within that animation.

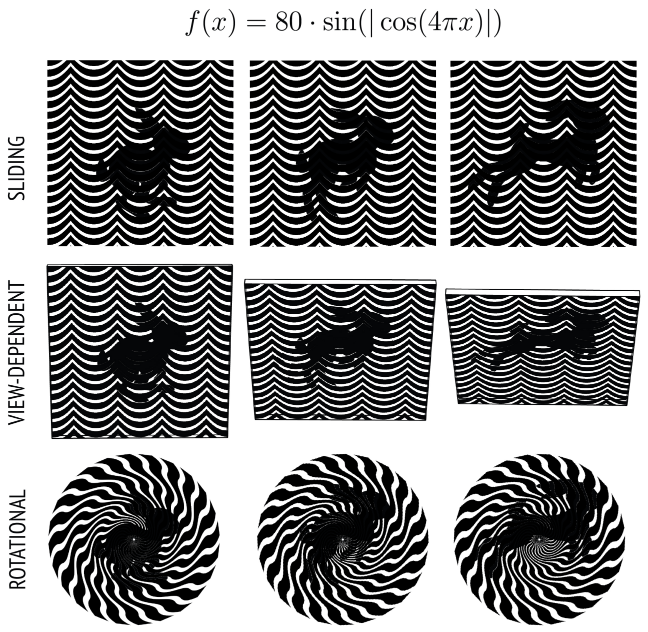

Figure 10: $f(x) = 80 \cdot \sin(|\cos(4\pi x)|)$ **produces coherent animations across variants. Each row shows the sequential animation frames produced by its respective method.**

Nested animations can encode a variety of interactions within a single interlaced image. For instance, we can apply linear interlacing to encode multiple linear animations (Fig. 11a), or even to encode radial animations, enabling playback that responds to both translation and rotation (Fig. 11b). Conversely, we can use radial interlacing to encode linear animations, producing sequences that require rotation to select an animation and translation to play it.

Theoretically, we can continue this nesting process indefinitely to create animations within animations within animations. But in practice, each nesting operation adds another barrier that partially occludes the underlying image and causes progressive loss of visual information. We quantify this trade-off in more detail in Section 8.

5 *FabObscura*: Towards Computationally Designing Barrier-Grid Animations

We developed *FabObscura* as a computational design tool that enables users to explore parts of our design space and construct different types of barrier-grid animations. Our tool has two main components: the **pattern editor** (Fig. 12a) which provides controls for specifying the animation's design parameters, and the **interactive canvas** (Fig. 12b) which provides visual feedback and direct interaction with the resulting animation.

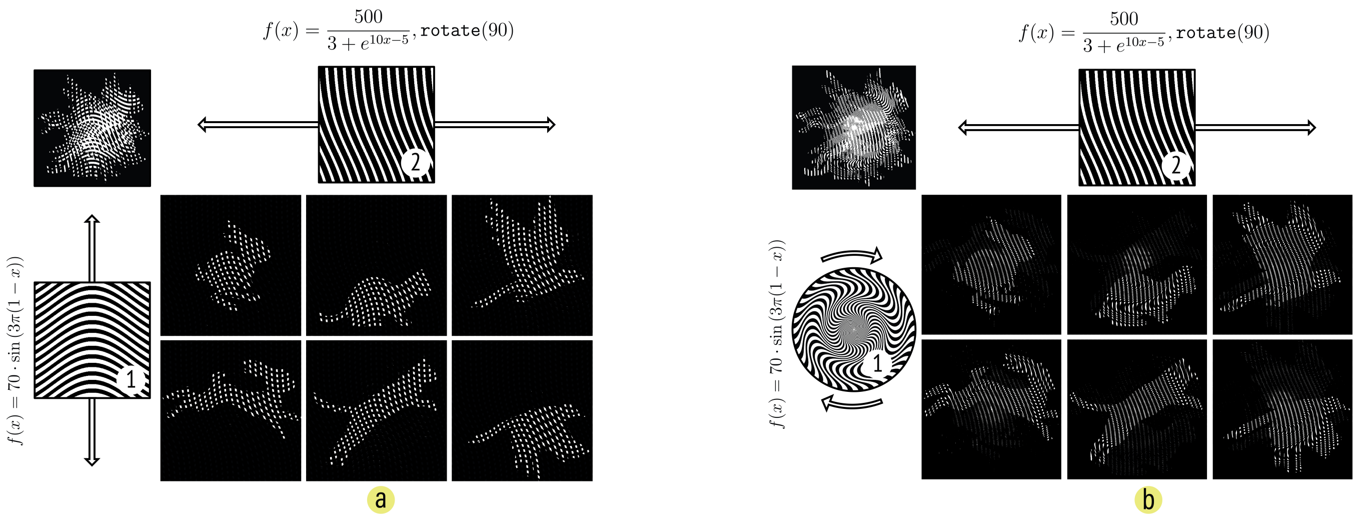
$f(x) = \frac{500}{3+e^{10x-5}}$, rotate(90)
$f(x) = 70 \cdot \sin(3\pi(1-x))$
1
2
a
b

Figure 11: This example illustrates two approaches for making a nested animation from three two-frame animations. In method (a), sliding barrier (1) vertically activates the inner animation sequence, while sliding barrier (2) horizontally selects which animation is being shown. In method (b), rotating the radial barrier (1) activates the inner animation sequence, while sliding barrier (2) horizontally selects the animation.

5.1 Designing Animations

Our tool uses the high expressive ceiling of our parameterization to provide an exploratory design process where users can construct valid animations from any mathematical function.

Consider Figure 12, which shows the main *FabObscura* interface. ① To begin, the designer selects their animation sequence, which is a folder containing an image sequence of animation frames. The animation sequence serves as the visual content revealed through the barrier pattern. ② As the designer enters the pattern unit function, ③ the interface updates a plot that visualizes the mathematical expression. This real-time feedback allows designers to develop intuition about how different functions map to visual outcomes. ④ Next, the designer can adjust the resolution of their animation. Lower resolution animations use coarser interlacing that requires larger movements to advance between frames, while higher resolution animations have finer interlacing that makes them more sensitive to movement. ⑤ Finally, the designer can specify the direction of motion by applying a rotation. ⑥ After confirming the pattern parameters, the system updates the interactive canvas with the animation and specified barrier pattern.

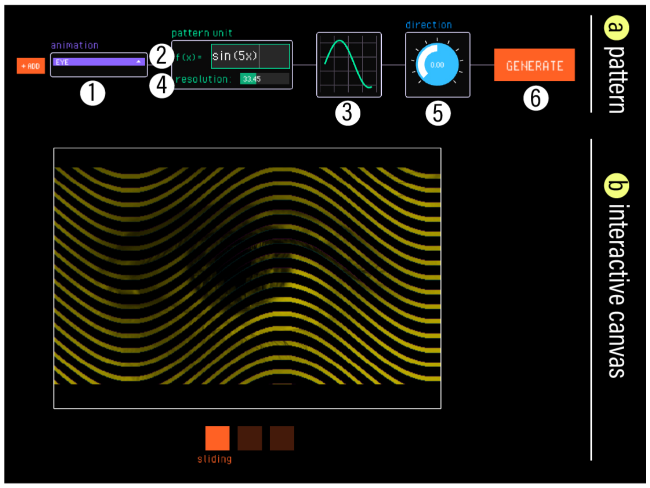

Figure 12: The *FabObscura* interface. In (a) the pattern editor, the designer configures their animation following our barrier-grid pattern parameterization. In (b) the interactive canvas, the designer interacts with their animation across different barrier-grid animation representations.

5.2 Interacting with Animations

The low viscosity of our parameterization allows our tool to provide an interactive canvas where designers can see how their animations perform under different barrier-grid configurations. It supports three interaction modes, each corresponding to a barrier-grid animation variant (Fig. 13).

Sliding. The **sliding interaction mode** creates animations that respond to sliding movements. *FabObscura* simulates this interaction by making the barrier track the cursor's y-coordinate (Fig. 13a). To ensure proper frame masking, the tool restricts the sliding interaction to follow the pattern's predefined direction of motion.

Viewpoint. The **viewpoint interaction mode** uses motion parallax to create view-dependent animations (Fig. 13b). In this mode, *FabObscura* simulates the physical separation between barrier and

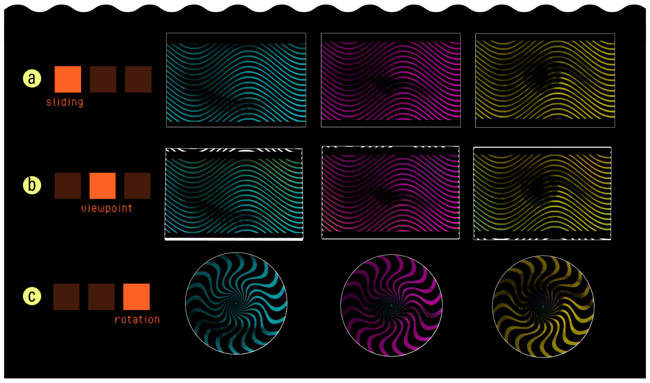

Figure 13: The *FabObscura* interface supports three interaction modes based on the established barrier-grid animation types: (a) *sliding*, (b) *viewpoint*, and (c) *rotation*.

interlaced image layers with a 3D preview. Designers can test different viewing angles by moving their cursor to virtually tilt the animation, and right-clicking locks the current position.

Since view-dependent animations operate in 3D space, the viewpoint interaction mode has additional sliders for adjusting the viewing distance from the camera, field of view, and distance between the barrier and interlaced layers. As the user adjusts these values, the tool corrects any misalignments that may arise from perspective by automatically resizes the interlaced image (which is further away from the camera) to align with the barrier.

Rotation. The **rotation interaction mode** creates animations that respond to rotational movements (Fig. 13c). *FabObscura* simulates rotational interactions by automatically turning the barrier based on the angle between its center and the cursor's position.

5.3 Experimenting with Nested Animations

While nested animations can seem initially counterintuitive, *FabObscura* provides an abstraction that helps designers build intuition for and experiment with these hierarchical structures. Clicking +Add creates a second set of pattern parameters for nesting, while -Del reverts to standard pattern generation.

Consider Figure 14, which shows the interface for creating nested animations. ① The designer selects two animations to nest. ② The top row of pattern parameters controls the settings for each individual animation within the nest, ③ while the bottom row defines the pattern that determines how users switch between these animations. ④ By default, the direction for the nesting barrier is set to be orthogonal to the animation direction, but the designer can adjust this value as desired. Once the designer confirms the parameters, the system creates a nested animation that uses a sliding barrier as the animation mask.

As with standard animations, the interactive canvas allows designers to interface with the nested animation under different interaction modes. Nested animations involve moving multiple barrier layers to access different levels of the nesting hierarchy. The designer can switch between the barrier being manipulated with the tab key (Fig. 15).

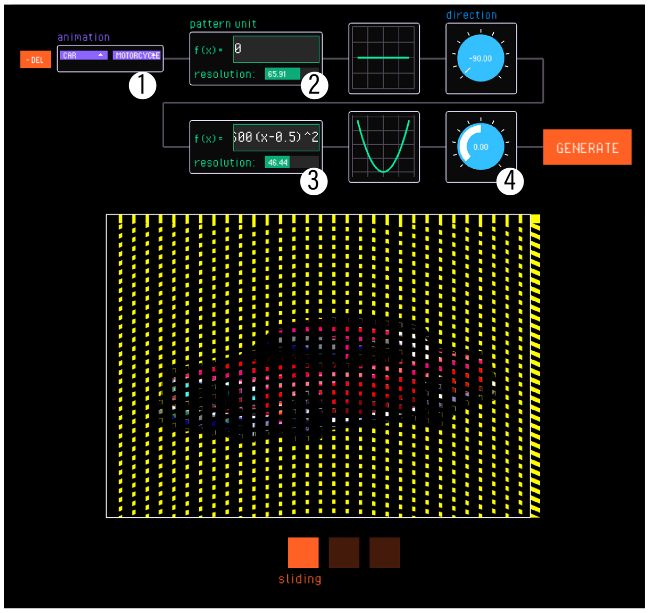

Figure 14: To create a nested animation, the designer selects two animations to nest and specifies the pattern parameters for each nesting layer.

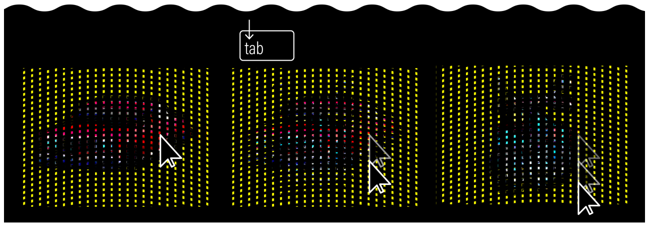

Figure 15: Designers access different levels of the nesting hierarchy by switching between the barrier being manipulated.

5.4 Outputs

Each time the user generates their animation, *FabObscura* outputs image files for each animation type. For standard animations, the system produces six raster images: one barrier pattern and one interlaced image for each of the three animation variants. For nested animations, the system produces one additional barrier that stores the nested pattern.

6 Applications

To showcase how *FabObscura* can extend the visual expressivity of physical objects, we use our system to fabricate a range of animations and explore potential applications.

6.1 Objects with Reconfigurable Appearances

Barrier-grid animations can be used to dynamically reconfigure an object's appearance to match changing contexts. To demonstrate this capability, we designed a reconfigurable coaster (Fig. 16) that accommodates to different beverage types. The coaster features three distinct designs: a coffee mug, a glass of water, and a cocktail glass. By sliding the coaster's outer sleeve, users can immediately change the displayed design to visually complement the beverage currently being served. For a contemporary aesthetic, we chose a modulated triangle wave function to produce a chevron-like pattern. This not only enhances the coaster's visual appeal but also demonstrates how barrier patterns themselves can serve as decorative elements that contribute to an object's design language.

$$f(x) = 65|4x \mod 2 - 1|,\ \texttt{rotate}(90)$$

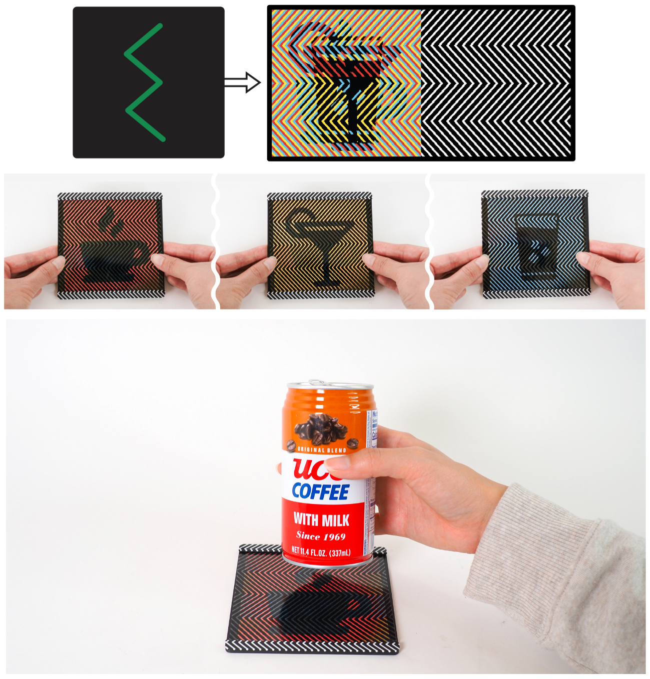

Figure 16: A coaster whose appearance can be reconfigured to complement its beverage.

6.2 Responsive Design

As barrier-grid animations rely on viewing angle and positional changes that occur naturally in everyday objects, we can use them to create responsive designs across diverse physical contexts.

Dynamic Containers. We designed two containers that animate as the user opens them by turning the lids counterclockwise. The first container depicts a three-frame animation of a snapping crocodile (Fig. 17). We stylistically emphasize the crocodile's action by interlacing it with a high frequency sine wave. The second container shows a three-frame animation of a flower gradually opening (Fig. 18). We create a soft spiraling effect that smoothly radiates out from the center by interlacing the frames with a radial function with constant curvature. Despite both using rotational motions, each container conveys a distinct aesthetic through its animation and interlacing pattern. These examples show how animation enhances visual communication in physical designs: a snapping crocodile for a jar of sharp thumb tacks, and a blooming flower for a box of delicate seeds.

$$f(r) = 30\sin(10\pi r^2)$$

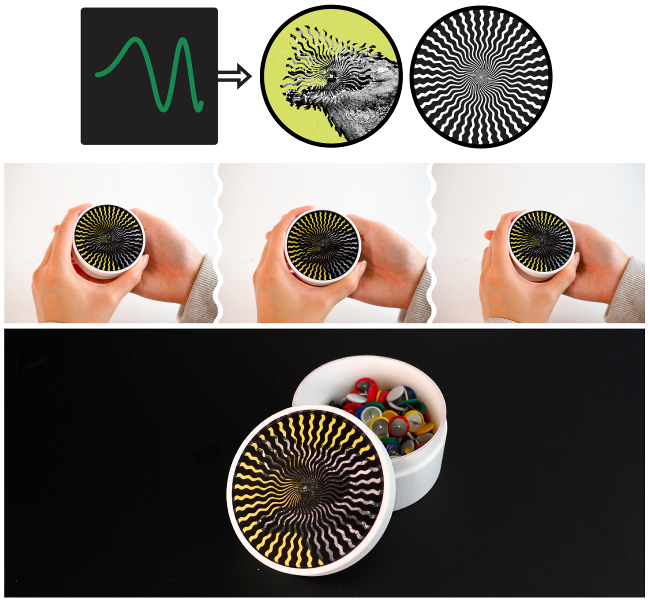

Figure 17: A thumb tack box with a snapping crocodile lid.

$$f(r) = 37$$

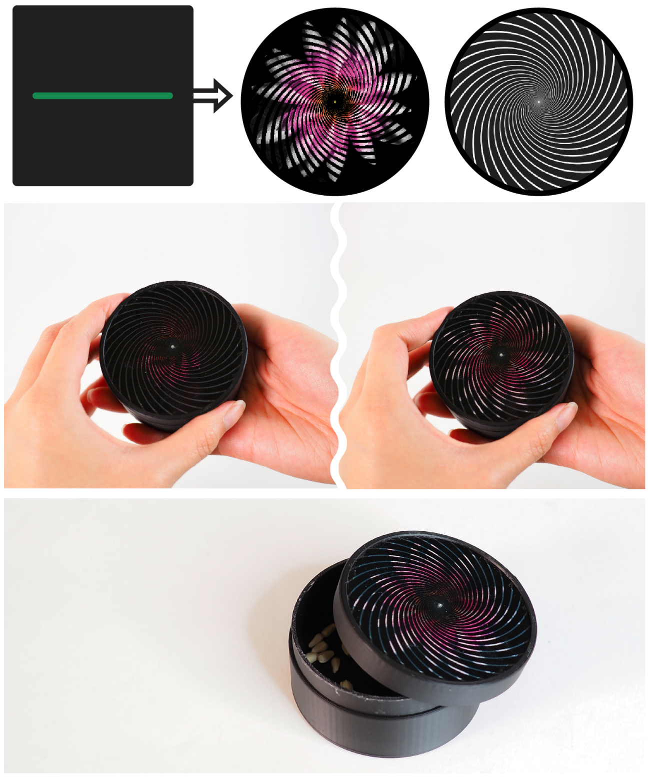

Figure 18: A seed box that blooms each time it is opened.

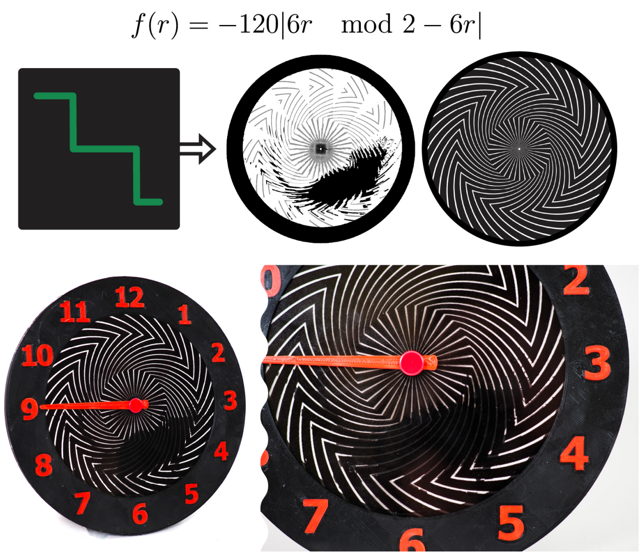

Figure 19: A clock design with a mouse that runs as it ticks.

Animated Clock Face. We made a clock face that holds a six-frame animation of a mouse on a wheel (Fig. 19). As the second hand moves, the mouse races against time and scurries along the walls of the clock. We applied a staircase function to create an arrow-like radial barrier pattern that emphasizes the clockwise motion. This example illustrates how *FabObscura* designs can work alongside common mechanisms, without compromising their functionality, by using shared interaction patterns.

6.3 Multimodal Interactions

Nested barrier-grid animations can support multimodal interactions that are not easily achievable with other passive display methods.

Low-Tech Interfaces. We explored how nested animations can be used as low-tech interfaces that mimic digital interactions. We designed an interactive display that nests two four-frame animations showing quarter-turn views of low-poly car and motorcycle 3D models (Fig. 20). Moving the barriers vertically switches between the two models, while moving horizontally reveals different angles of the selected vehicle. This interaction mimics digital 3D model exploration interfaces, where horizontal dragging orbits around objects and vertical scrolling switches to different views.

Reconfigurable View-Dependent Signage. Our second nested animation combines a view-dependent animation with a sliding barrier to create a reconfigurable sign that animates as viewers walk past it (Fig. 21). We nested two six-frame animations: one displaying "Wildlife Viewing Area" alongside a flying pelican, and another showing "Beware of Crocodiles" beside a slinking crocodile. Vertically sliding the nested barrier pattern switches between the message being displayed. For the inner barrier that controls the underlying animations, we used a piecewise function that bows outward in the middle while remaining straight at the edges. This directs visual interest toward the center where the animals are moving, while maintaining text legibility in the peripheral regions.

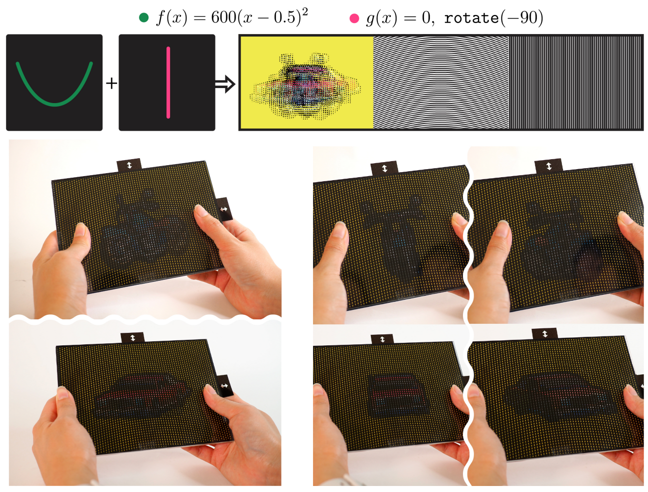

Figure 20: An interactive low-tech display that encodes quarter-turn views for two 3D models.

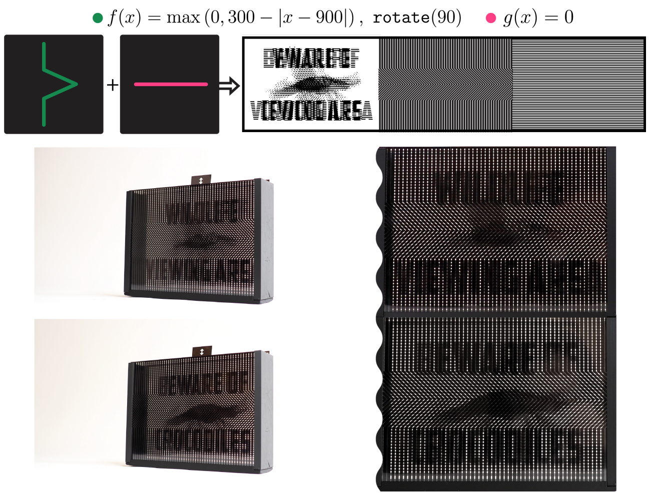

Figure 21: A reconfigurable dynamic sign that animates as people walk it. Users can choose which message is being displayed by sliding the inner barrier.

7 Implementation

We use the Processing graphics library to build the *FabObscura* design tool. We additionally run a Python Flask server in the background to receive data from Processing, construct animations, and output images.

7.1 Visualization

Our Python program receives values corresponding to the animation parameters from Processing and outputs raster images corresponding to the interlaced image and barrier for the linear and rotational animations. Processing displays the animation by drawing the barrier on top of the interlaced image.

Since view-dependent animations operate in 3D space, we visualize them by applying a standard perspective projection with a user-defined viewing distance (ranging from 20 to 80 inches). We additionally use a 16-degree field of view.

7.2 Constructing Linear Animations

Recall that a linear barrier-grid animation is constructed from an interlaced image I and barrier pattern B that moves along a given direction of motion θ. We rotate the set of animation frames so that the vertical direction aligns to the direction of motion (see below). Then, for each frame of width w and height h, both I and B are $w \times h$ images that comprise $\lfloor \frac{h}{t} \rfloor$ strips with thickness t. Each strip follows a path defined by the pattern unit function f.

Creating the interlaced image and barrier. Suppose we have a sequence of n animation frames, $F_1, \ldots, F_n$.To establish notation, if M is a image divided into rows of strips with thickness t, we let $M[i]$ denote the i-th row of image M.

Given a pattern unit function f, direction of motion θ, and interlacing thickness t, we construct the interlaced image I as follows:

(1) As a preprocessing step, we first rotate each input frame F_j by angle θ (where $j = 1, \ldots, n$). We will assume the rotated frames are of size $w \times h$.
(2) For each rotated frame F_j, segment it into $N = \lfloor \frac{h}{t} \rfloor$ strips of thickness t, displaced vertically column-by-column to the path defined by f.
(3) Initialize the interlaced image I as a $w \times h$ image, then interleave strips from different input frames according to:

$$I[k] = F_{k \bmod n}[k], \qquad k \in \{1, \ldots, N\}.$$

This expression ensures that adjacent strips in I come from consecutive frames in the animation sequence.
(4) Finally, we rotate I back by angle $-\theta$ to ensure that all its strips are oriented along the direction θ.

We construct the corresponding barrier B as follows:

(1) Initialize B as a $w \times h$ image and rotate it by angle θ.
(2) Segment B into N strips of thickness t, following the same paths defined by f.
(3) Construct barrier B by coloring its strips such that:

$$B[k] = \begin{cases} \texttt{white} & \text{if } k \bmod n = 0 \\ \texttt{black} & \text{otherwise} \end{cases}, \quad k \in \{1, \ldots, N\}.$$

This produces a pattern that alternates between a white (transparent) strip of thickness t and $n - 1$ black (opaque) strips of thickness t. The pattern precisely aligns with I to reveal only one frame at a time.
(4) Rotate B back by angle $-\theta$ to ensure that all its strips are oriented along the direction θ.

Perspective correction (for view-dependent animations). To correct any misalignment that may occur due to viewing distance and layer spacing, we perform an additional resizing step on I to ensure that it lines up with B in perspective.

Our perspective correction works as follows: Given a specified viewing distance, field of view, and spacing between layers, we use Processing's raytracer to render a frontal view of both layer planes (without textures). We then pass this image to OpenCV in Python—applying thresholding, identifying contours, and calculating the bounding boxes of both layers. By comparing the width of the inner rectangle (back layer) to the outer rectangle (front layer), we determine the precise scaling ratio needed to compensate for perspective distortion. We apply this ratio to scale the interlaced image to match the front barrier layer.

7.3 Constructing Rotational Animations

Recall that a rotational barrier-grid animation is constructed from a radially symmetric image I and a corresponding barrier B, which rotates clockwise around its center. For animation frames of width w and height h, both I and B are circular $r \times r$ images where $r = \min(w, h)$. These images are divided into $\left\lfloor \frac{360}{\gamma} \right\rfloor$ wedge-shaped segments, separated by radially curved boundaries spaced at angular intervals of γ degrees. Each boundary follows a curved path defined by the pattern unit function f.

Determining the shape of radially curved boundaries. Since f is originally defined in Cartesian space, we repurpose it to control how radial paths bend in polar space, so that they curve outward without intersecting their neighbors. To do so, we adapt a method for generating radial moiré patterns by Gabrielyan [7]. We generate each radially curved boundary by tracing a path outward from the origin, where the angular position $\theta(\rho)$ as a function of radius ρ evolves according to a local inclination angle $f(\rho)$.

Let β represent the desired angle between neighboring boundary curves and let $N = \left\lfloor \frac{360}{\beta} \right\rfloor$. To generate the k-th boundary curve (where $k \in \{0, 1, \ldots, N-1\}$), we initialize it with the starting angle:

$$\theta_0^{(k)} = k\beta$$

We then trace the curve by iterating over a sequence of radii $\rho_0 = 0, \rho_1, \ldots, \rho_m$, which represent evenly spaced distances from the center of the circle to its outer edge. At each step, the angular position evolves according to the recurrence:

$$\theta_{i+1}^{(k)} = \theta_i^{(k)} + \frac{180}{\pi} \cdot \frac{(\rho_{i+1} - \rho_i) \cdot \tan(f(\rho_i))}{\rho_i}, \quad i \in \{0, \ldots, m-1\}.$$

This process defines the shape of the k-th radially curved boundary as it spirals outward from its base angle $\theta_0^{(k)}$, without intersecting neighboring boundaries. Repeating this process for all $k \in \{0, 1, \ldots, N-1\}$ produces a seamless division of a circle into N wedge-shaped segments with radially curved boundaries.

Creating the interlaced image and barrier. Suppose we have a sequence of n animation frames, $F_1, \ldots, F_n$, each of size $w \times h$. For notation, if M is a polar image of radius R, centered at the origin and divided into wedge-shaped segments, with each separated by angle γ, then let $M[i]$ denote the i-th wedge segment in polar space.

Given a pattern unit function f and interlacing thickness t, we construct the radially interlaced image I as follows:

(1) As a preprocessing step, we crop each animation frame F_j (for $j = 1, \ldots, n$) to a circular region with diameter $r = \min(w, h)$. We then convert each cropped frame to polar coordinates by mapping its Cartesian $r \times r$ image to a polar representation with radius R. We choose the polar angle β to be equal to the interlacing thickness t.

(2) For each polar frame F_j, segment it into $N = \left\lfloor \frac{360}{\beta} \right\rfloor$ wedges bounded by radially curved paths defined by f.
(3) Initialize the interlaced image I as a polar image of radius R. Construct I by interleaving wedge segments from the input frames as follows:

$$I[k] = F_{k \bmod n}[k], \qquad k \in \{1, \ldots, N\}.$$

This ensures that adjacent wedges in I originate from consecutive frames in the sequence.

We construct the corresponding radial barrier B as follows:

(1) Initialize B as a polar image of radius R, centered at the origin.
(2) Segment B into N wedges bounded by the same radially curved paths defined by f.
(3) Construct B by assigning color to each wedge segment:

$$B[k] = \begin{cases} \texttt{white} & \text{if } k \bmod n = 0 \\ \texttt{black} & \text{otherwise} \end{cases}, \qquad k \in \{1, \ldots, N\}.$$

This creates a radial mask that reveals one frame at a time during rotation, while occluding the others.

7.4 Nesting Animations

To create a nested animation that encodes m animations with n frames each, we first interlace each animation according to their barrier parameters as explained in the previous section. Since these images were interlaced with the same function, they also share a common barrier. We treat the resulting m interlaced images as animation frames and interlace them with a second set of interlacing parameters to obtain the nested barrier. For deeper levels of nesting, we follow the same process to interlace the resulting nested images.

8 Technical Evaluation

We conduct a technical evaluation to assess how hyperparameter selection affects the visual quality of barrier-grid animations. Since barrier-grid animations rely on self-occlusion, our goal is to gather data at sufficient scale to richly detail the tradeoffs between expressivity and visual quality introduced by different barrier pattern densities. In doing so, we aim to offer concrete guidance for designing high-quality barrier-grid animations and shed light on the strengths and limitations of our framework.

Our evaluation focuses on straight line patterns as representative cases that isolate key parameters of our framework. As fundamental building blocks widely used in practice, straight line patterns provide generalizable insights that extend to more complex pattern functions. Specifically, we compare basic "slat" barrier patterns that support motion in one direction with nested "lattice" patterns that support bidirectional interactions. These patterns serve as canonical examples that allow us to measure the effects of resolution, spacing, and axis configuration without confounding variables.

8.1 Datasets

We perform technical evaluations on a dataset containing 30 sets of frames, where each frame is a 600×600 image. Within this, there are 15 animations and 15 sets of black and white icon images, which reflect the two primary ways to apply our method: frame-by-frame animations and encoding multiple distinct images of any kind (not necessarily from an animation) into the same surface.

Animations. Given the challenge of finding datasets with expressive motion in a few frames, we selected 15 example animations, each consisting of between 2 and 10 frames, sourced from Adobe Stock and converted to grayscale.

Icons. The 15 sets of icons test the ability of our approach to encode a wide variety of images with no inherent overlap in their black and white regions. This dataset consists of 1,000 distinct vector graphic icons from the Font Awesome solid icon pack, providing a robust test of the method's generalizability across diverse visual inputs.

8.2 Metrics

We assess the performance of our approach in simulation using three error metrics to compare the rendered output frame R to the desired input frame I.

MSE Loss. The Mean-Squared-Error (MSE) measures average squared difference between the pixels of R and I, quantifying how much of the desired image is visible or obscured by barriers. While MSE is a good measure of pixel-wise accuracy, it falls short as a measure of human perceptual error, as human perception involves far more complex processes than simply analyzing individual pixels [41].

SSIM Loss. The Structural Similarity Index (SSIM) evaluates the similarity between R and I, focusing on structural information, contrast, and luminance, which aligns more closely with human perception than does MSE. It is calculated as a product of three factors: the linear correlation between images, an index of consistency of mean luminance levels across images, and an index of consistency of contrast levels across images [40]. SSIM values range from -1 to 1, where 1 indicates perfect structural similarity.

Perceptual (VGG) Loss. To better align with human visual perception, we compute the VGG loss using the VGG-19 neural network, comparing the feature maps extracted from images R and I at different layers of the network. These feature maps capture high-level information, such as textures, patterns, and object shapes, which are more representative of how humans perceive images compared to simple pixel-based differences. We measure VGG loss as

$$\text{VGG Loss} = \sum_{l} \|\phi_l(I) - \phi_l(R)\|_2^2,$$

where ϕ_l represents the feature map extracted from layer l of the VGG-19 model. This loss quantifies differences in visual quality that pixel-wise metrics like MSE often miss. We use the feature maps from specific convolutional layers after max-pooling operations, often referred to as "style layers," with indices $L \in \{0, 5, 10, 19, 28\}$ [35]. These layers focus on capturing higher-level abstractions, such as texture and object recognition, which are crucial for measuring perceptual quality.

8.3 Experimental Setup

For our technical evaluation, we calculate error metrics on raytraced results generated by a custom Python raytracing program. Our evaluation uses view-dependent animations. To accurately reveal the desired frame of interest in our evaluation, we incrementally adjust the viewing direction. Specifically, we calculate the necessary angle for each adjustment such that the viewing direction shifts by exactly one slat per step, using principles of similar triangles to determine the correct angles.

Figure 22 compares the outputs of our raytracer with a real fabricated sample. We see strong correspondence between the simulated and physical results, validating our rendering approach as a reliable proxy for evaluating fabricated designs. Note that the black square visible at the bottom of the real samples is an artifact caused by the holding point during photography.

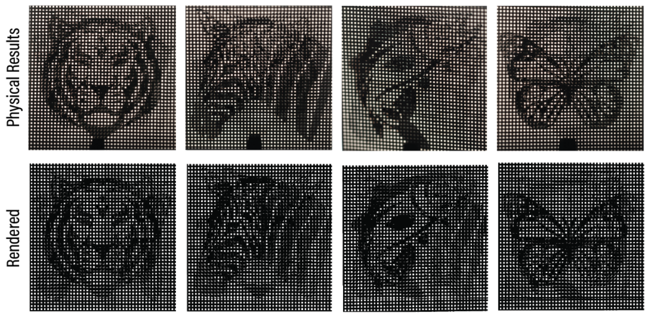

Figure 22: Top: Four photographs of a fabricated lattice surface from four different viewing angles, taken from a distance of 20 inches. Bottom: Rendered results from our Python raytracer from the same viewing angles, using a simulated viewing distance of 20 inches.

8.4 Results

Comparing Lattices to Slats. We begin by testing and comparing the slats approach to the lattice grids approach. We fix the number of frames for our test cases to 4, our barrier spacing (i.e., the thickness of transparent units) to 3 pixels, and the front and back plane distance to 0.25 inches. We then use both slats and lattices to produce a four-frame animation for each subset of our dataset (30 subsets total), where the frames are vertically offset for slats and displayed in a grid for lattices. Table 1 reports the mean, median, standard deviation, lowest, and highest of each of our error metrics.

Table 1: Comparison of slats and lattices across different error metrics.

Metric	Approach	Mean	Median	Std Dev	Lowest	Highest
MSE	Slats	0.4214	0.4496	0.1448	0.0388	0.6752
	Lattice	0.4334	0.4516	0.1438	0.0386	0.6889
SSIM	Slats	0.2884	0.2896	0.2029	0.0011	0.7133
	Lattice	0.1848	0.1867	0.1376	0.0006	0.4846
VGG Loss	Slats	92.71	81.78	39.68	29.36	228.71
	Lattice	95.43	86.73	38.89	25.04	230.81

We find that lattices and slats exhibit similar performance, although lattices consistently have higher mean MSE and VGG Loss, and a lower SSIM score. This is intuitively what we would expect, as with the lattice approach more of the resulting rendered images are obstructed due to having twice as many lines in the barrier than with the slats approach. Therefore, while lattices provide more flexibility for handling multiple axes of view at once, there is a trade-off between this flexibility and the quality of results.

Effect of Barrier Spacing on Animation Quality. Next, we investigate how varying the barrier spacing influences the quality of our animations. Using our dataset, we run the process for barrier spacings of n pixels, for $n \in \{1, \ldots, 10\}$.

Figure 23 visualizes VGG loss as a function of barrier spacing for both slats and lattices. For lower barrier spacings (1-4 pixels), lattices exhibit high variance, but their overall performance is comparable to that of the slats. However, as barrier spacing increases and the bars become thicker, the performance gap between the two approaches widens significantly, with slats achieving nearly half the VGG loss of lattices. This suggests that slats are more robust in maintaining quality as barrier spacings increase. Across all barrier spacings, slats consistently also outperform lattices in terms of lower MSE and higher SSIM. On average, slats achieve an MSE of 0.4082 and SSIM of 0.3236, compared to lattices with an average MSE of 0.4223 and SSIM of 0.2536, with the performance gap widening as the spacing increases.

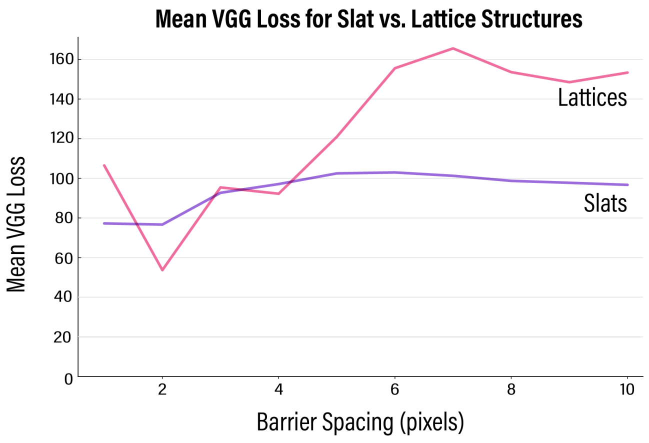

Figure 23: A comparison of VGG loss across different barrier spacings (in pixels) for lattice and slat patterns. Mean VGG loss increases significantly with higher barrier spacing for lattices, while slats maintain a more consistent rate across different spacings.

Effect of Changing the Frame Count. We further analyze the impact of varying the number of frames in our test cases. Using a fixed barrier spacing of 3 pixels, we evaluate two types of lattice barriers: one that combines four frames arranged in a 2×2 grid (i.e., nesting two two-frame animations), and another that combines nine frames arranged in a 3×3 grid (i.e., nesting three three-frame animations). We exclude animations with fewer than nine frames from this study.

We additionally evaluate image quality for each configuration using our established metrics and present them in Table 2. Our findings indicate that the 3×3 configuration does not result in a significant degradation in image quality compared to the 2×2 setup.

Effectiveness of Perspective Correction. To mitigate ghosting artifacts in view-dependent animations, our method rescales the interlaced image to maintain proper alignment with the barrier from a specified viewing distance and layer separation.

Table 2: Comparison of 2×2 and 3×3 approaches across different error metrics.

Metric	Approach	Mean	Median	Std Dev	Lowest	Highest
MSE	2×2	0.4441	0.4583	0.1289	0.2194	0.6889
	3×3	0.5196	0.5425	0.1505	0.1776	0.8121
SSIM	2×2	0.2079	0.2017	0.1350	0.0030	0.4846
	3×3	0.2306	0.2211	0.1456	0.0028	0.5328
VGG Loss	2×2	86.27	85.20	11.97	61.45	109.19
	3×3	73.82	73.16	11.23	46.76	112.31

Figures 24 and 25 qualitatively assess the impact of our perspective correction method with our renderer. The bottom half of each plot highlights how perspective correction enhances frame clarity and minimizes ghosting artifacts. These improvements are particularly pronounced at shorter rendering distances and in structures with greater distances between layers.

9 Discussion

By understanding how barrier-grid animations can be constructed through a unified mathematical framework, *FabObscura* enables more expressive and complex animations than previously possible. This section discusses the design insights emerging from our technical evaluation, acknowledges the limitations of our approach, and outlines directions for future work.

9.1 Design Best Practices

Guided by our technical evaluation, we propose the following best practices for constructing high-quality barrier-grid animations:

(1) **Use high contrast designs with distinct silhouettes.** High-contrast designs with distinct silhouettes are versatile because they are recognizable even as visual fidelity decreases. This makes them well-suited for a variety of constructions, including nested configurations that tend to exhibit lower fidelity than their non-nested counterparts.
(2) **Use moderate barrier spacings for nested configurations.** Nested configurations exhibit high variance at lower barrier spacings and degrading performance at higher spacings. In singly-nested configurations, our evaluation shows that the optimal spacing is roughly 0.5-0.8% of the design's width.
(3) **For view-dependent animations, apply perspective correction for close viewing or thick structures.** While it is less critical when the viewing distance is large or the layers are closely spaced, perspective correction becomes especially important at close range or when there is significant separation between layers.

9.2 Limitations and Future Work

While *FabObscura* provides a foundation for designing barrier-grid animations, the current prototype of the system does not exhaustively cover the design space of our parameterization. For instance, the interface can only nest two animations even though it is algorithmically possible to create more complex constructions. Future implementations could support deeper nesting hierarchies while providing visualization tools to help users negotiate the design trade-offs that accompany each additional layer.

One part of the design space that we do not explore in-depth is the unit function representation. For example, rather than assuming that patterns have a fixed resolution throughout, we can make the resolution vary depending on the pattern segment—or, instead of assuming that unit functions are continuous lines, we can depict them as a set of discrete shapes (Fig. 26). This opens possibilities for more expressive pattern designs that cannot easily be described by continuous functions alone. Future work could investigate other representations for pattern units and explore their visual and interactive affordances.

Finally, while our technical evaluation provides valuable quantitative insights into animation quality, it does not capture how hyperparameter choices may affect user interaction. A comprehensive user study that qualitatively assesses the perceptual and interactive impact of varying hyperparameters is a promising direction for future work.

10 Conclusion

We presented *FabObscura*, a process for systematically designing novel interactive barrier-grid animations. By recontextualizing barrier-grid animations as parameterized mathematical functions, *FabObscura* preserves the canonical properties of barrier-grid animations while giving them infinite expressivity.

One might wonder why it is important to revisit early art forms like barrier-grid animations—after all, since their inception, there have been centuries of developments in animation, display technologies, and dynamic materials. But in the same way that early film techniques continue to inform modern cinematography, barrier-grid animations lay the foundation for modern processes such as lenticular printing and autostereoscopic displays. Perhaps by providing a process for making barrier-grid animations more expressive, we can also inspire new approaches for the technologies that come after them.

Acknowledgments

We thank Josh Pollock, Dylan Wootton, and Matt Beaudouin-Lafon for reviewing paper drafts and providing valuable feedback. We also thank Lyvia Kenfack for their help with early prototypes of the system.

References

[1] Lubna Abu Rmaileh and Alan Brunton. 2023. Meso-Facets for Goniochromatic 3D Printing. *ACM Transactions on Graphics (TOG)* 42, 4 (2023), 1–12.
[2] JA Arriaga-Hernández and A Jaramillo-Núñez. 2018. Ronchi and Moiré Patterns for Testing Spherical and Aspherical Surfaces Using Deflectometry. *Applied Optics* 57, 34 (2018), 9963–9971.
[3] Amit Bermano, Ilya Baran, Marc Alexa, and Wojciech Matusk. 2012. Shadowpix: Multiple Images From Self Shadowing. In *Computer Graphics Forum*, Vol. 31. Wiley Online Library, 593–602.
[4] Jimmy Chion. [n. d.]. Circular Moiré Slit Animation. http://www.instructables.com/Circular-Moir%C3%A9-Slit-Animation/
[5] Stelian Coros, Bernhard Thomaszewski, Gioacchino Noris, Shinjiro Sueda, Moira Forberg, Robert W Sumner, Wojciech Matusik, and Bernd Bickel. 2013. Computational design of mechanical characters. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–12.
[6] Jenna Didier. [n. d.]. *The Spring*. https://www.jennadidier.com/the-spring-aka-swimming-with-sharks-dedicated-in-hollywood/
[7] Emin Gabrielyan. 2007. Fast optical indicator created with multi-ring moiré patterns. (2007).
[8] Thomas RG Green. 1989. Cognitive dimensions of notations. *People and computers V* (1989), 443–460.

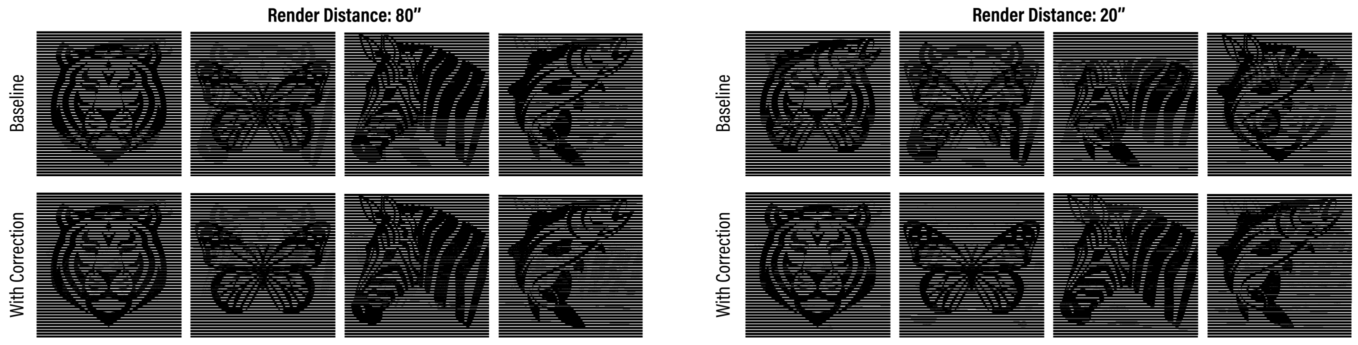

Figure 24: Left: A four-frame animation without perspective correction (top) and with correction (bottom), viewed from a distance of 80 inches. Right: the same animation viewed from 20 inches, where ghosting and artifacts are more prominent without correction. All animations measure 2×2 inches, with a distance of 0.25 inches between the interlaced and back barrier layers.

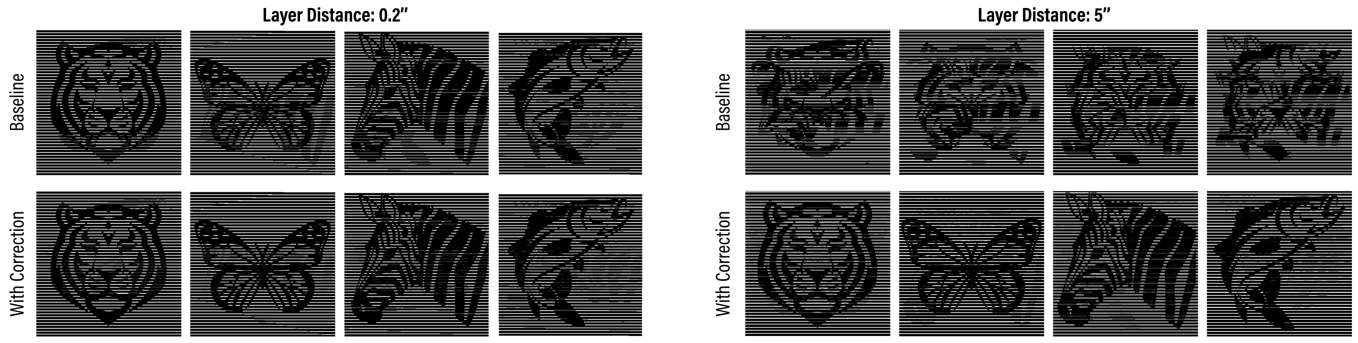

Figure 25: Left: A four frame animation without perspective correction (top), with a distance of 0.2 inches between the interlaced (back) and barrier (front) layers. and 5. Right: the same animation with a distance of 5 inches between the interlaced and barrier layers, where ghosting and artifacts are more prominent without correction. All animations measure 2×2 inches and are viewed from a distance of 80 inches.

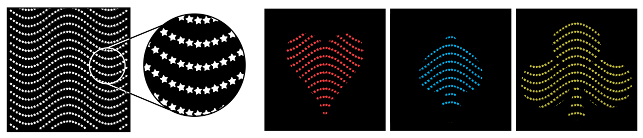

Figure 26: Beyond continuous functions, we can interlace images with discrete functions to create distinctive effects. Here, we use a discretized sinusoidal function with star-shaped points to encode a three-frame animation.

[9] Ollie Hanton, Mike Fraser, and Anne Roudaut. 2024. DisplayFab: The State of the Art and a Roadmap in the Personal Fabrication of Free-Form Displays Using Active Materials and Additive Manufacturing. In *Proceedings of the CHI Conference on Human Factors in Computing Systems*. 1–24.

[10] Ollie Hanton, Zichao Shen, Mike Fraser, and Anne Roudaut. 2022. FabricatINK: Personal Fabrication of Bespoke Displays Using Electronic Ink From Upcycled E Readers. In *Proceedings of the 2022 CHI Conference on Human Factors in Computing Systems*. 1–15.

[11] Stephen Herbert. 2010. The Optically Animated Artwork of Rufus Butler Seder. Online. http://www.rufuslifetiles.com/TheOpticallyAnimatedArtworkofRufusButlerSeder.pdf

[12] Yuhua Jin, Isabel Qamar, Michael Wessely, and Stefanie Mueller. 2020. Photochromeleon: Re-programmable multi-color textures using photochromic dyes. In *ACM SIGGRAPH 2020 Emerging Technologies*. 1–2.

[13] Sarah Anne Kushner, Paul H Dietz, and Alec Jacobson. 2022. Interactive 3D Zoetrope with a Strobing Flashlight. In *Adjunct Proceedings of the 35th Annual ACM Symposium on User Interface Software and Technology*. 1–3.

[14] Anat Levin, Daniel Glasner, Ying Xiong, Frédo Durand, William Freeman, Wojciech Matusik, and Todd Zickler. 2013. Fabricating BRDFs at High Spatial Resolution Using Wave Optics. *ACM Transactions on Graphics (TOG)* 32, 4 (2013), 1–14.

[15] Jiaji Li, Shuyue Feng, Maxine Perroni-Scharf, Yujia Liu, Emily Guan, Guanyun Wang, and Stefanie Mueller. 2025. Xstrings: 3D Printing Cable-Driven Mechanism for Actuation, Deformation, and Manipulation. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems (CHI '25)*. Association for Computing Machinery, New York, NY, USA, Article 6, 17 pages. https://doi.org/10.1145/3706598.3714282

[16] Jiaji Li, Mingming Li, Junzhe Ji, Deying Pan, Yitao Fan, Kuangqi Zhu, Yue Yang, Zihan Yan, Lingyun Sun, Ye Tao, and Guanyun Wang. 2023. All-in-One Print: Designing and 3D Printing Dynamic Objects Using Kinematic Mechanism Without Assembly. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems* (Hamburg, Germany) *(CHI '23)*. Association for Computing Machinery, New York, NY, USA, Article 689, 15 pages. https://doi.org/10.1145/3544548.3581440

[17] Mightool. [n. d.]. Scanimation. http://www.mightool.com/scanimation/

[18] Leo Miyashita, Kota Ishihara, Yoshihiro Watanabe, and Masatoshi Ishikawa. 2016. Zoematrope: A system for physical material design. In *ACM SIGGRAPH 2016 emerging technologies*. 1–1.

[19] Brad Myers, Scott E Hudson, and Randy Pausch. 2000. Past, present, and future of user interface software tools. *ACM Transactions on Computer-Human Interaction (TOCHI)* 7, 1 (2000), 3–28.

[20] Wakasa Noguchi and Hiroki Nishino. 2022. High-Low Tech Ombro-Cinéma. In *ACM SIGGRAPH 2022 Posters*. 1–2.

[21] Colin Ord. 2007. *Magic Moving Images: Animated Optical Illusions*. Tarquin Publications, London.

[22] Maxine Perroni-Scharf and Szymon Rusinkiewicz. 2023. Constructing Printable Surfaces with View-Dependent Appearance. In *ACM SIGGRAPH 2023 Conference Proceedings*. 1–10.

[23] Petar Pjanic and Roger D Hersch. 2015. Color Changing Effects With Anisotropic Halftone Prints on Metal. *ACM Transactions on Graphics (TOG)* 34, 6 (2015), 1–12.

[24] Rechenraum. [n. d.]. Animbar. http://animbar.mnim.org/

[25] 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.

[26] Gianni A. Sarcone. [n. d.]. Official Website of Gianni A. Sarcone. https://www.giannisarcone.com/

[27] Gianni A. Sarcone. 2014. Rotating Kinegrams. https://www.behance.net/gallery/17609995/Rotating-Kinegrams Accessed: 2025-03-11.

[28] Gianni A. Sarcone. n.d.. Kinegrams, Art in Motion. Online. http://giannisarcone.com/Kinegrams.html From Sarcone's Studio – A Sarcone & Waeber Web Resource.

[29] Rufus Butler Seder. 2001. Visual Display Device With Continuous Animation. https://patents.google.com/patent/US6286873B1/en U.S. Patent No. 6,286,873. Filed August 26, 1999, and issued September 11, 2001.

[30] Rufus Butler Seder. 2007. *Gallop!: A Scanimation Picture Book*. Workman Publishing Company, New York.

[31] Ticha Sethapakdi, Laura Huang, Vivian Hsinyueh Chan, Lung-Pan Cheng, Fernando Fuzinatto Dall'Agnol, Mackenzie Leake, and Stefanie Mueller. 2023. Polagons: Designing and Fabricating Polarized Light Mosaics with User-Defined Color-Changing Behaviors. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–14.

[32] Ticha Sethapakdi, Mackenzie Leake, Catalina Monsalve Rodriguez, Miranda J Cai, and Stefanie Mueller. 2022. KineCAM: An Instant Camera for Animated Photographs. *Proceedings of the ACM on Computer Graphics and Interactive Techniques* 5, 4 (2022), 1–9.

[33] Ticha Sethapakdi, Paris Myers, Tianyu Yu, Juliana Covarrubias, Mackenzie Leake, and Stefanie Mueller. 2024. Thermochromorph: Dynamic Relief Printing with Thermochromic Inks. In *SIGGRAPH Asia 2024 Art Papers (SA '24)*. Association for Computing Machinery, New York, NY, USA, Article 11, 7 pages. https://doi.org/10.1145/3680530.3695445

[34] Pengfei Shen, Rui-Zeng Li, Beibei Wang, Ligang Liu, and Tao Zhuang. 2023. Scratch-based Reflection Art via Differentiable Rendering. *ACM Trans. Graph.* 42, 4 (2023), 65–1.

[35] Karen Simonyan and Andrew Zisserman. 2014. Very Deep Convolutional Networks for Large-scale Image Recognition. *arXiv preprint arXiv:1409.1556* (2014).

[36] 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.

[37] Motiform Studio. [n. d.]. Human Walking Scanimation Clock Face. https://makerworld.com/en/models/905999-human-walking-scanimation-clock-face#profileId-866116

[38] F.J. Vernay. 1898. *The Motograph Moving Picture Book*. Bliss, Sands, London.

[39] Robin A Walker. 2013. Holograms as Teaching Agents. In *Journal of Physics: Conference Series*, Vol. 415. IOP Publishing, 012076.

[40] Zhou Wang, A.C. Bovik, H.R. Sheikh, and E.P. Simoncelli. 2004. Image Quality Assessment: From Error Visibility to Structural Similarity. In *IEEE Transactions on Image Processing*, Vol. 13. 600–612.

[41] Zhou Wang and Alan C Bovik. 2009. Mean Squared Error: Love It or Leave It? A New Look at Signal Fidelity Measures. *IEEE signal processing magazine* 26, 1 (2009), 98–117.

[42] Jennifer Weiler. 2020. *Beyond Plastic Filament: An Exploration of 3D Printing as a Part of Creative Practices*. Ph. D. Dissertation. Arizona State University.

[43] Gordon Wetzstein, Douglas Lanman, Wolfgang Heidrich, and Ramesh Raskar. 2011. Layered 3D: Tomographic Image Synthesis for Attenuation-based Light Field and High Dynamic Range Displays. In *ACM SIGGRAPH 2011 papers*. 1–12.

[44] Takumi Yamamoto and Yuta Sugiura. 2023. Turning Carpets Into Multi-image Switchable Displays. *Computers & Graphics* 111 (2023), 190–198.

[45] Zeyu Yan, Hsuanling Lee, Liang He, and Huaishu Peng. 2023. 3D Printing Magnetophoretic Displays. In *Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology*. 1–12.

[46] Takegi Yoshimoto, Shuto Murakami, and Homei Miyashita. 2023. Edible Lenticular Lens Design System. In *Adjunct Proceedings of the 36th Annual ACM Symposium on User Interface Software and Technology* (San Francisco, CA, USA) *(UIST '23 Adjunct)*. Association for Computing Machinery, New York, NY, USA, Article 21, 3 pages. https://doi.org/10.1145/3586182.3616656

[47] Qing Yu, Bao-min Li, and Qi-yun Wang. 2024. The Effectiveness of 3D Holographic Technology on Students' Learning Performance: A Meta-analysis. *Interactive Learning Environments* 32, 5 (2024), 1629–1641.

[48] Tianyu Yu, Weiye Xu, Haiqing Xu, Guanhong Liu, Chang Liu, Guanyun Wang, and Haipeng Mi. 2023. Thermotion: Design and Fabrication of Thermofluidic Composites for Animation Effects on Object Surfaces. In *Proceedings of the 2023 CHI Conference on Human Factors in Computing Systems*. 1–19.

[49] Daniel Campos Zamora, Mustafa Doga Dogan, Alexa F Siu, Eunyee Koh, and Chang Xiao. 2024. MoiréWidgets: High-Precision, Passive Tangible Interfaces via Moiré Effect.. In *CHI*. 329–1.

[50] 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*. 1184–1196.

[51] Yunyi Zhu, Cedric Honnet, Yixiao Kang, Junyi Zhu, Angelina J Zheng, Kyle Heinz, Grace Tang, Luca Musk, Michael Wessely, and Stefanie Mueller. 2024. PortaChrome: A Portable Contact Light Source for Integrated Re-Programmable Multi-Color Textures. In *Proceedings of the 37th Annual ACM Symposium on User Interface Software and Technology*. 1–13.