filtered.ink: Creating Dynamic Illustrations with SVG Filters

Tongyu Zhou
Brown University
Providence, Rhode Island, USA
tongyu_zhou@brown.edu

Connie Liu
Brown University
Providence, Rhode Island, USA
connie_liu1@brown.edu

Jeff Huang
Brown University
Providence, Rhode Island, USA
jeff_huang@brown.edu

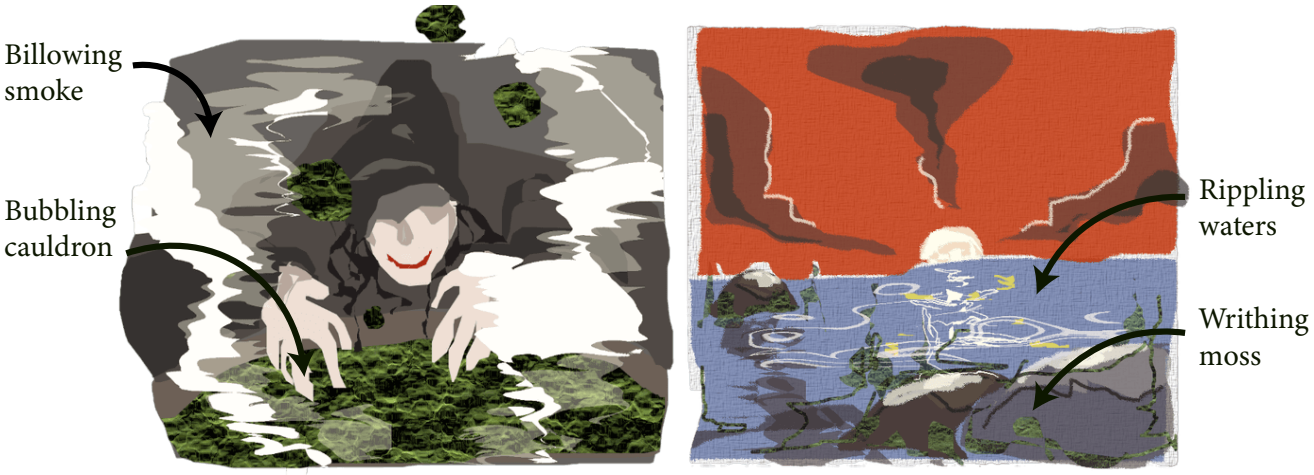

Figure 1: Example dynamic illustrations created using filtered.ink, with animated parts annotated above.

ABSTRACT

Vector illustrations are object-based, allowing them to be scaled, queried, and compressed without loss of quality. However, they are not widely adopted due to the lack of visual expressiveness compared to raster illustrations. Scalable vector graphic (SVG) filters, raster transformations applied to vector primitives, can potentially fill this niche, but are hindered by a steep learning curve and a coding prerequisite. I demonstrate the efficacy of using SVG filters for raster-like vector illustration by creating a real-time visual editor for users to design and share filters to use as brushes. By examining interactions that occur when crafting, remixing, and using filters through a task-based usability study, I identify workflow patterns and new avenues of expression unique to this framework. I then synthesize insights that can be used to design future iterations of vector-based art applications.

CCS CONCEPTS

• **Human-centered computing** → **User interface toolkits**; **Web-based interaction**.

CHI '22, April 23 – April 28 2022, Hamburg, Germany

ACM ISBN 978-1-4503-XXXX-X/18/06…$15.00
https://doi.org/10.1145/1122445.1122456

ACM Reference Format:
Tongyu Zhou, Connie Liu, and Jeff Huang. 2023. filtered.ink: Creating Dynamic Illustrations with SVG Filters. In *Proceedings of CHI '23: ACM CHI Conference on Human Factors in Computing Systems (CHI '22).* ACM, New York, NY, USA, 12 pages. https://doi.org/10.1145/1122445.1122456

1 INTRODUCTION

Still images capture moments in time. When their static backgrounds are juxtaposed against select animated motifs, however, they are able to capture *impressions* of time. This style of media, considered cinemagraphs in photography [19] and kinetic textures in illustration [20, 21], allows artists to additionally control the propagation of elements over time and lends to new creative possibilities. GIFs and other video formats are commonly used to publish this type of media. While raster formats have long dominated the web, which has in return been optimized for them, there is still a lot of space for new content that does not scale or suit the pixel-based paradigm. Especially with the increasing density of screen resolutions and displays, raster media created on smaller, older screens will appear pixelated when projected onto larger, newer ones. To solve this problem, vector-based media such as SVGs have become more commonly used due to their scalability, compactness, and browser compatibility.

SVGs are not the dominant image format, however, especially in illustration. This is partially due to their comparative lack of expressiveness and steeper learning curve. Although there are existing tools such as Adobe Illustrator [16] or Inkscape [43] that enable SVG drawing, achieving the same level of detailed textures comparable to raster-based visual effects via vectors may require significant coding knowledge or GPU acceleration to render efficiently [7]. In

addition, creating fluid animations for SVGs is a non-trivial task for the user [4]. Existing methods to create animated raster-like effects require dedicated online tutorials, which add friction to an originally fluid artistic process and deters casual creators.

To explore the best modalities for creating SVG-based animated paintings, we introduce *filtered.ink*, a web-based platform for painterly SVG creation and manipulation. To match the expressiveness of raster, we incorporate SVG filters, or raster transformations that operate on existing vector primitives and are re-rendered at each display resolution, as integral components of the creation workflow. The platform provides several different ways to create, manipulate, and remix these filter transformations that can then be applied onto strokes on a canvas. To understand how SVG filters fit into the visual artist's natural workflow and affect illustration outcomes, we conduct a usability study with hobbyist and professional artists who are familiar with common vector-based design tools like Illustrator, but not with SVG filter transformations. By identifying key behavioral patterns and preferences shared amongst the artists when using a new creative tool with a technical learning curve, we derive further insights into how parameter-based tools for vector art can be designed to better support the artist's mental model.

This work identifies the affordances needed to support vector-based animated textured illustrations and the creative possibilities this new framework can introduce. This is achieved through 1) creating a system that enables animated SVG painting and 2) a study that analyzes patterns and identifies design guidelines for SVG-based brush authoring.

2 BACKGROUND AND RELATED WORK

Figure 2: Left: An SVG image of a trump card without any filters applied. Middle: The SVG image with a filter combining feTurbulence, feDiffuseLighting, and feComposite primitives applied to the background to generate a paper-like texture. Right: The SVG image with a filter combining feTurbulence and feDisplacement primitives applied to the diamonds to generate a "scattering" effect.

Our system focuses on SVG filters, a series of graphical transformations that can be applied to SVGs to return bit-mapped results. The SVG filter output differs from that of traditional raster graphics as it is recomputed at each rendering resolution and thus retains high fidelity. Within the SVG specification, it is defined with a `<filter>` tag, after which multiple SVG filter primitives with fine-tunable parameter values can be sequentially combined to imitate realistic textures or artistic styles (see Figure 2). Multiple filters can also be defined within the same SVG file and differentiated between via the `id` attribute. These filters can then be applied to SVG elements, which we will refer to as *strokes* to accommodate the context of drawing, by adding an attribute `filter="url(#name-of-filter)"` to the SVG element. Currently, all major browsers support SVG filters.

Our system design is informed by current strategies for vector creation and manipulation and uses themes from tools for node-based authoring and dynamic textures in illustration.

2.1 Vector Creation and Manipulation

Existing systems that aid users in vector-based media creation or manipulation focus primarily on vector primitives that combine to formulate strokes. These systems fall into two general categories: machine-based automatic generation or human-centered interactions to guide the authoring process. Examples in the former category include SVG-VAE [29], Im2Vec [34], Cloud2Curve [6], and Stylized Neural Painting [46], all of which rely on learned models to perform style propagation and vector reconstruction from an original raster graphic. For reconstruction of messier raster sketches that produce small spaces between sketched lines, Delaunay subdivision has been used for automatically coloring sub-regions [33]. Other methods operator solely within the vector space; DeepSVG [4] focuses on animating existing SVGs instead by training a model that outputs smooth interpolations between one SVG and another to facilitate seamless transitions between the two. Other strategies that expand the toolkit for vector fabrication forgo the SVG format altogether and engineer new ones tailored for expressing creativity. For example, temporal diffusion curves (TDC) [28] were introduced as a new type of vector graphic that captures the evolution of strokes to achieve a wider variety of styles and effects for vectorized painting. Strategies that map a user's mouse location to triangle mesh representations [2] have also been proposed to allow users to paint with vectors without dealing with the underlying implementation.

These auto-generative approaches either 1) excel in imitation but not in freedom of expression or 2) suggest new specifications suitable for the intended domain but has limited accessibility due to lack of browser support. Thus, other systems have instead shifted focus onto easing the authoring experience for vectors through novel interactions, focusing on SVGs as the vector graphic standard on the web. Sketch-n-Sketch, an SVG editor, was introduced to show how direct manipulation interfaces with general-purpose programs as the representation format can allow both experts and non-experts to use programmable systems [11]. This editor was later further extended to incorporate a paradigm where users can both manipulate code to affect the rendered graphic and directly tweak the graphic to see the reflected changes in code, which the authors termed output-directed programming [12]. This bidirectional strategy reconciles flexibility and ease of use.

Since these SVG-based systems focus on vectorized primitives, they do not support a wide variety of textures and styles enabled by new formats such as TDC. Our work thus fills this gap by facilitating the creation of raster-like transformations on SVGs through filters.

2.2 Node-based Authoring

The SVG filter specification is a procedural material graph. Inverse procedural material modeling has been used to estimate graph parameter values that can best reconstruct an input RGB image. Some of these methods such as Hu et. al's pipeline [14] or MATch [39] train neural networks that require searching through pre-existing

procedural node graphs to find a best match, while more recent strategies focus on semi-automatic generation to bypass the large data set requirement [15]. The resultant node graphs can be further fine-tuned by a user to change material properties such as roughness, color palette, or shininess.

Editing the generated materials and textures via a node graph is similar to the block-based approach in visual programming, which has been used in systems such as Scratch [35] and Blockly [9] to help novices understand the semantic connections between program state and output. In comparison to standard text-based programming, this block-based approach has been demonstrated to incite greater learning gains and levels of interest in programming [45]. The visual metaphor of directly manipulating blocks is also particularly well suited for programming environments where the output is also visual. For example, Para [18] gives the user live manual control of declarative constraints to enable non-linear editing to produce complex procedural art. Similarly, Demystified Dynamic Brushes [26] links state to visual outputs so that users can control program execution through direct selection.

While our work is similar to these block-based systems by providing similar controls to produce visual outputs, we further enable greater dimensions for expression by providing the ability to animate program states.

2.3 Dynamic Textures in Illustration

Motion in illustration can convey unique contextualized messages and emotions via dynamic visuals. Previous studies have attempted to understand and identify patterns in how they can do so by introducing design spaces for animated narratives [40] to provide useful guidelines for future animations. Other studies focus on authoring tools to create these dynamic effects. For example, Draco [21] demonstrated the appeal of illustrations with kinetic textures and showed that continuous animation effects on standalone elements juxtaposed with still backgrounds are helpful to designers in creating ambient motions that are not disruptive. Kitty [20] pushes this further by introducing an underlying graph model to capture visual, spatial, and temporal relationships between drawn objects, and Druid [22] allows users to remix a set of motion amplifiers, based on preexisting 2D animation principles, for rapid iteration.

While the previous systems provided effective frameworks for general animated illustrations, the variety of scenarios during which illustration can be created in also demands situational solutions. In vector-based illustrations, for example, re-scaling the animation to smaller screen sizes may result in visual crowding if all details are maintained. Re-targeting algorithms have thus been introduced to preserves spatial detail on resize and appropriately redistribute it based on importance [38]. Other systems emphasize subjective style; while temporal flickering may seem like an undesirable artifact to some, adding *controllable* temporal flickering can help convey emotional expressivity by allowing digital animation to mimic traditional, hand-colored ones [8]. In cases where the user is less familiar with animation techniques, crowd workers have also been motivated to create re-mixable interactive behaviors easily and accurately at 2.9 minutes on average [25].

In these systems, the artist creates animations on top of originally static images. Our work instead treats animation as a first-class object. Since animation can be applied as a filter transformation to a stroke, we introduce a new workflow experience where users can "draw animations" directly.

3 FILTERED.INK

3.1 Design Considerations

Informed by prior work, our vector-based drawing tool aims to preserve the same visual expressiveness as that of raster graphics while maintaining benefits of the vector format, including scalability, compactness, query-ability, and lightweight animation. Our system is built using React and Node.js and is guided by two Shneiderman design principles for general creativity support tools [41]: 1) support exploratory search and 2) enable collaboration. Based on these ideas, we developed specific goals for SVG drawing, targeting experienced artists but coding novices.

3.1.1 Support exploration of SVG filters for drawing. Filter fabrication in existing vector drawing software such as Adobe Illustrator or Inkscape involves either manually typing out the SVG filter code or tuning filter primitive parameters. Neither support filter animation. To understand the benefits and challenges of both, we ran a pilot study on an earlier iteration of the system that employed both the manual code entry and parameter tuning strategies with 3 casual digital artists. All three users preferred parameter tuning, but still felt that the learning curve was too high. One user commented that filter-editing was "not visual enough" and thus was confusing to understand. This comment aligned with insights from previous work that revealed developing efficient workflows that aligned with how artists *naturally worked* can help them develop self-sufficiency [17, 27]. Our final system thus incorporates visual abstraction for parameter tuning and animation to better accommodate the artist's mental model.

3.1.2 Enable collaboration through sharing and remix. Repeated exposure to other examples has been demonstrated to improve the quality of creative work [24, 42]. Remix, defined as the process of taking existing artifacts created by someone else to combine or manipulate into new cultural blends [23], has also been commonly used to generate new art. While there is a fine line between remix and plagiarism, particularly when the sense of ownership is strong [10], remixing SVG filters that are then applied to create original drawings can depersonalize the action and lighten the sense of ownership. By supporting the sharing and remix of SVG filters, our system exposes users to the ideas of others and provides a source of inspiration. Another aspect of collaboration is the awareness of others and the ability to hand-off your work to other stakeholders [32]. To support this, we implement quality-of-life features for users to share and import both the filters and entire drawings that can be decomposed into strokes and filters.

With these goals in mind, we further use filtered.ink to explore two research questions, focusing on the workflow of users during *dynamic vector painting*, differentiating it from that of existing vector art, and the potential visual styles this medium can achieve:

Q1. What is the natural workflow that users follow to create, remix, or use SVG filters in dynamic vector painting and how can we support this?

Q2. How does the distribution of filters in the final composition inform potential styles of dynamic vector painting?

3.2 The SVG Filter Editor

The filter editor uses visual abstraction to separate different components of SVG code into distinct, recognizable modules, as demonstrated in Figure 3, to encourage user exploration. These modules are A) the SVG filter primitives, B) the primitive parameters, and C) the animations that can be applied to each parameter. On this editor, the user can also see a live preview of the filter they are currently designing, as well as delete or share the filter.

3.2.1 Primitives. Visual effects in SVGs can be created by combining 17 possible filter primitives: *feBlend*, *feColorMatrix*, *feComponentTransfer*, *feComposite*, *feConvolveMatrix*, *feDiffuseLighting*, *feDisplacementMap*, *feDropShadow*, *feFlood*, *feGaussianBlur*, *feImage*, *feMerge*, *feMorphology*, *feOffset*, *feSpecularLighting*, *feTile*, and *feTurbulence* [31]. Since our tool focuses on drawing specifically, for simplification we do not include *feImage* since it maps an input image to an output, *feOffset* since it only shifts an input by some (x, y) offset (which can be achieved by drawing in a different location on the canvas), *feMerge* since similar effects can be achieved by modifying filter order, and *feTile* since it only tiles smaller inputs to an output. For the remaining filter primitives, we use a node-graph representation to depict the relationships between each filter in the composition, as illustrated in A of Figure 3. Each node represents a filter primitive, while a directed edge from some node A to a node B indicates that the result of primitive A is fed into primitive B as its input. Users can add, delete, and move edges to modify how the filters interact with each other.

Some filter primitives, such as *feTurbulence* and *feFlood*, do not actually take inputs. In this case, whenever these primitives are added, their effects override those of the previous primitives. To emphasize this, we highlight the active primitives in black and leave the overridden primitives in white. The user can easily switch to different active "branches" by clicking on nodes in overridden branches.

3.2.2 Primitive Parameters. Upon clicking on a node, the parameters of that primitive are rendered on the left-hand side of the node-graph, as depicted in B of Figure 3. To better complement the workflow of the artist, we abstract away the actual code of the filter in place of visual representations. An example of one such abstraction can be seen in the figure, where the edge weights of a bipartite graph are used to represent matrix values in the *values* parameter of the *feColorMatrix* filter. The values in *feColorMatrix* are used to apply a transformation matrix that converts each existing RGBA value to new ones, as shown below.

$$feColorMatrix_{values} = \begin{bmatrix} r_R & r_G & r_B & r_A & r_C \\ g_R & g_G & g_B & g_A & g_C \\ b_R & b_G & b_B & b_A & b_C \\ a_R & a_G & a_B & a_A & a_C \end{bmatrix} \tag{1}$$

$$\rightarrow \begin{bmatrix} r_R & r_G & r_B & r_A & r_C \\ g_R & g_G & g_B & g_A & g_C \\ b_R & b_G & b_B & b_A & b_C \\ a_R & a_G & a_B & a_A & a_C \\ 0 & 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \\ A \\ 1 \end{bmatrix} = \begin{bmatrix} R' \\ G' \\ B' \\ A' \\ 1 \end{bmatrix} \tag{2}$$

Thus, each edge e_{ij} that connects node i to node j represents j_i in the matrix. Visually, the more opaque each edge is, the more contribution one color channel in the original color scheme contributes to the color channels in the new color.

3.2.3 Animation. Under the hood, we rely on W3C recommended SMIL 3.0 for filter parameter animations. Using this specification requires the user to provide the parameter to animate, the duration of the animation, and the values the animation cycles through. For simplicity, we restrict the animation to infinite loops through user-determined values. This is visualized in C of Figure 3. A pie-chart with n states represents the total number of values to cycle through for that parameter; clicking on each state allows the user to update parameter values using the visual representation. The clock hand that cycles through states on the pie-chart with speed dependent on the animation duration for one completed cycle is synced to animation in the live preview.

3.3 The Drawing Interface

After users finish designing a filter, they can either directly start drawing with the filter applied as a brush or apply the filter to an existing SVG element on the drawing interface. The drawing interface, depicted in Figure 4, features a toolbar with common drawing tools such as pen, fill, color dropper, move, undo, redo, erase, import, and download, in addition to SVG-specific tools such as change filter. Users can draw on the canvas below the toolbar with their mouse or tablet, or by holding down the `D` key and moving their cursor. The sidebar on the left features the currently active SVG filters and the metadata of the most recently drawn or currently selected brush stroke.

We enable cursor drawing by tracking mouse coordinates and mapping them to SVG polyline points. If a transformation is selected on the sidebar, we additionally pre-apply it to the stroke so that the effects can be seen as the stroke is being drawn. Each time a new stroke, or polyline, is drawn, we store the coordinates, color, width, opacity, time the stroke started, time the stroke ended, its associated SVG filter transformation ID, and a unique creator ID. To optimize rendering and reduce the number of coordinates displayed, we also simplify the polyline using a combination of the Douglas–Peucker and Radial Distance algorithms [1].

3.4 Remix

Users can share filters that they created to a global database, stored using Firebase, by clicking on the "Upload Filter to Presets" button in the filter editor. Other users on the application can then browse through these "preset" filters, import them, and remix them for

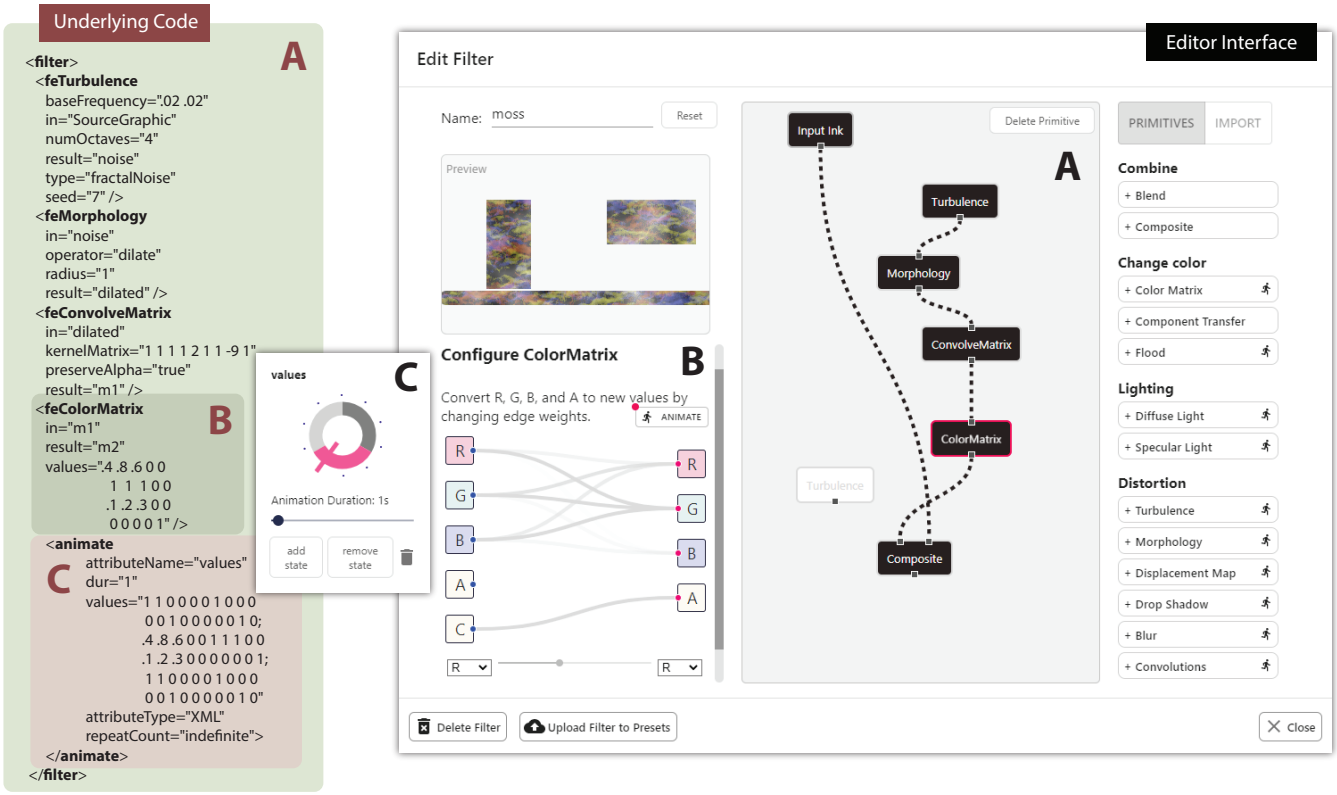

Figure 3: The editor interface uses A) a node-graph representation to depict the connection between filter primitive inputs and outputs, B) visual abstraction to illustrate the operations performed by the filter, and C) a pie chart representing animation states for parameter values. In this example, values in the *feColorMatrix* filter are represented as edge weights in a bipartite graph.

personal use. Examples of different filters created by users can be seen in Figure 5.

4 USAGE EVALUATION

We conducted a usage evaluation for filtered.ink to 1) understand the natural workflow of how users create animated vector paintings and 2) assess the application's conceptual clarity, usability, unique capabilities, and limitations. By examining the interactions that occur when users are drawing live with their own crafted filters as brushes, we can derive insights into designing future iterations of vector-based painting applications.

4.1 Participants

Nine participants took part of the study (6 female, 1 male, 2 non-binary), aged 20 to 22 years old (average 21). All participants had been creating art at some capacity since childhood, three of them were studying illustration in university, and all of them actively produced digital artwork for either a publication, professional practice, schoolwork, or recreation. Four participants had experience with coding, while five had introductory or no experience at all. Participants were recruited from the researcher's own artistic network, as well as from illustration teams from digital campus publications. The participants were all students from two neighboring universities, one of which focuses on art. Eight of the nine participants listed Procreate, Photoshop, and Illustrator as the applications they currently use to make illustrations, and every participant listed PNGs as their usual file format for their artwork. No participants had any prior experience or knowledge about SVG filters. Each participant received a cash compensation of $25 for their participation.

4.2 Study Protocol

All of the experiments were conducted using a Huion H420 Pen Tablet, mechanical keyboard, and 1920x1080 monitor. Each participant was asked to bring into the study an existing illustration they'd made on another digital platform to encourage user focus on the creative medium instead of devoting attention to the subject matter of the illustration. Informed by prior work that found learning instances to be greater in specific tasks in comparison to open-ended exploration [37], we constrained the participants to specific tasks that asked them to recreate existing filters and illustrations (rather than creating illustrations from scratch) to focus on participants' abilities to understand primitives and integrate filters into their existing workflow and art styles. The evaluation period lasted for 80 minutes for each participant, and consisted of the following steps.

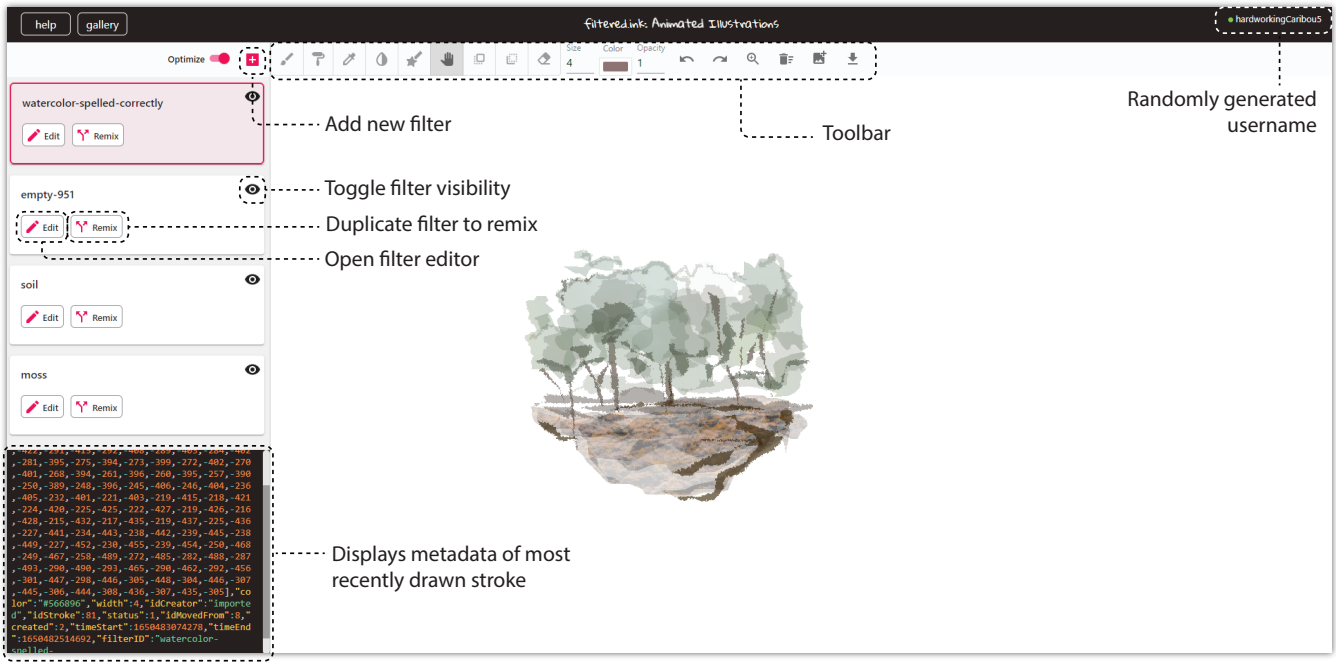

Figure 4: The drawing interface features a toolbar with common drawing tools, the canvas, a list of active SVG filters, and metadata about the most recently drawn stroke.

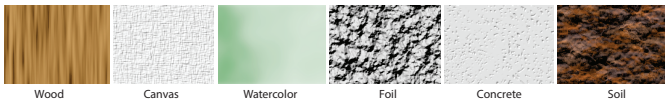

Figure 5: Examples of filters that user crafted using filtered.ink and shared with others.

4.2.1 Introduction (5~10 minutes). In the pre-survey, participants filled out a background questionnaire about their experience level with art, coding, and vector graphics/art/files. After the completion of the pre-survey, each participant was given a brief overview and demonstration of the application. The instructor walked each participant through the interface and the functionality available on the platform, demonstrating steps such as selecting preset filters, creating a new filter, adding primitives, and customizing effects. Participants were encouraged to try out the interface for themselves and get comfortable with the setup, as well as narrate their thought processes out loud.

4.2.2 Warm-up (15~20 minutes). For the first task, participants were asked to recreate an existing preset filter from the list of premade filters. The exercise covered the identification of a desired effect, the creation of a filter from scratch, the procedures and learnability of adding primitives, customization, and feasibility of achieving a desired effect. Participants were prompted with the preview of the target preset filter on the preset dropdown list as well as with their set of active filters on the filter sidebar, which they could refer to and utilize at any moment. They were prohibited from entering the preset filter's editor to view how the preset had been created. The facilitator did not intervene unless the participant had trouble using the system, and a time limit of 20 minutes was imposed. The purpose of this task was to observe whether the participants could easily reproduce a target effect.

4.2.3 Free-form Drawing (35~45 minutes). For the second task, participants were asked to recreate the illustration that they'd brought in using filtered.ink. Participants were encouraged to utilize and integrate filters into their workflow and artwork in whatever way they saw fit or was intuitive to them. Once done with their artwork, participants were asked to fill out a post-study questionnaire to provide feedback about the application. After the questionnaire, we conducted a semi-structured interview to understand the artists' thought processes and their overall experience of the system.

5 RESULTS

5.1 Supporting Natural User Workflow

We record interaction logs throughout each session to map out a new user's natural creative workflow, as depicted in Figure 6. In this plot, "drawing" interactions refer to instances whenever a participant drew a stroke on the canvas, while "filter editing" interactions refer to instances whenever a participant changed a filter parameter value or edited a filter animation. Filter editing primarily occurred in the beginning during the warm-up task. However, when the task concluded, most participants continued to edit filters during the free-form drawing task, where they had more freedom to work in ways more intuitive to them. After an uninterrupted filter editing phase, participants changed their workflow to focus on

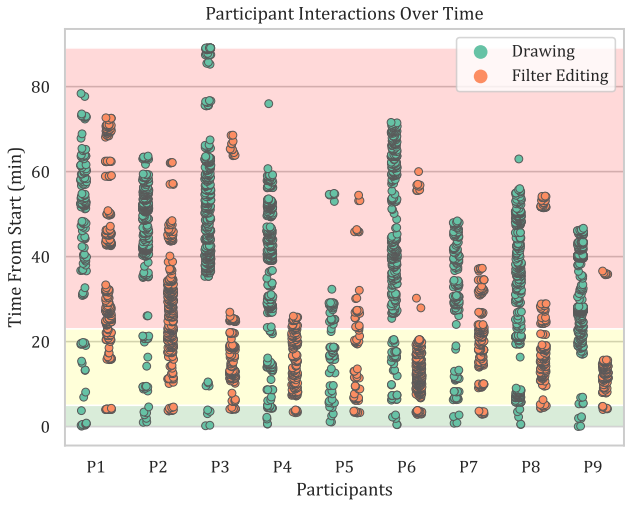

Figure 6: A plot showing instances where participants were drawing and editing the filters. The colored background regions refer to the average time taken for each task (green: introduction, yellow: exercise task, red: replication and animation). Filter editing generally occurred in a large phase near the beginning, after a bit of drawing but before a large amount of drawing. But filters were lightly edited during the drawing for most participants.

predominantly drawing on the canvas. However, most participants still return to the editor to make short bursts, or clusters, of filter modifications.

After each drawing session, we conduct a semi-structured interview to understand how filtered.ink can support and augment natural workflows. We group user responses based on Palani et. al's Creative Practitioners' Value Framework [32], focusing on three dimensions that creatives look for when adopting new tools that have been emphasized in prior literature of key design principles of creativity support tools [36, 41]. These dimensions are 1) the tool's features and functionality, its 2) original workflow integration, and the 3) interface and user experience. Italicized phrases in the following subsections refer to features that creative practitioners explicitly mention to be important to that dimension [32].

5.1.1 Features and Functionality. A tool's features and functionality refer to its set of abstractions that achieve a certain goal. In our case, users modify parameters in node-graphs to create animated brushes with which they can draw with. Participants reacted positively to the *dynamic responsiveness* of the node-graph in the filter editor, as in most cases modifying parameters values resulted in immediate visual updates in the filter preview.

Seven of the nine participants stated they could see themselves creating something new with the tool that they wouldn't be able to using previous systems, referring to the application's *specialization*, or ability to do unique tasks. Some focused on "the ability to texture vector images. Because the current process is so tiring, you don't see that around in a lot of vector art. This changes that" (P7), while others focused on the ability to animate the textures, "I think just the SVG filters are very unique and cool. I think I already use a lot of Photoshop to manipulate effects; to be able to see that in a dynamic drawing is cool" (P1). Several participants also stated how filtered.ink would open up the world of vector art for them. P5 mentioned that "It's interesting that it's a very painterly tool which isn't the case with other vector tools." Overall, participants with a primarily illustration background found the application to be an accessible introduction into creating vector art, with P4 noting that "I think adding an animated aspect to my existing illustrations would be cool. I want more vector art in my portfolio, and this would be really cool step. I want to make more vector art because when it comes to branding and design, vectors are really important since they're scalable too."

Two participants also expressed a lot of interest in the *collaboration* aspect of the "Import SVG" function, expressing that this would be really beneficial to current digital creators and communities that already work a lot with the SVG format. P4 noted the system's *generalizability*, "Graphic designers would get a kick out of this, especially those whose work is experimental. Right now this platform is geared towards illustrators, but graphic designers who work with SVGs would get a kick out of adding SVG filters to their work." P5 affirmed this with their excitement about potential applications in their own practice of illustrative typography, saying "instead of having to export each animated letterform as a GIF, having an animated SVG would save much more space and be infinitely resizeable for every publishing platform." P7 also mentioned the potential of animated SVG's to be "great for use in texturing for 3D models and movie applications. I can't stop thinking about how it would be a great resource for games because it's infinitely scalable and has math-based animation, so you could recycle the same asset without losses."

5.1.2 Workflow Integration. Creative systems that can integrate with artist workflows should co-exist and work together with the affordances of existing tools. Participants relied heavily on processes and features that matched prior creativity support tools: P7 specifically stated that "I'm imparting my knowledge of Photoshop and Procreate onto this" when navigating both the filter editor and the toolbar on the drawing canvas. Participants also pointed out their desire for additional functionality that have become standard on most other popular drawing platforms such as layers, resizing, polygonal shape tools, and a more robust selection tool. The lack of these small functionalities was cited as the biggest inhibitor to them adopting filtered.ink into their existing workflow, as participants wanted the familiar controls that are provided on other digital creation platforms.

For the first task, all participants relied on blind experimentation when attempting to understand what each primitive could do and how to achieve their desired effects. Participants fell into two categories of filter integration in their workflow. Some participants first drew the base of their illustration with standard non-filtered brushes, and then applied filters afterwards either with filter brushes or by using the "Change Filter" tool to convert existing non-filtered strokes into filtered strokes. Other participants drew directly with the filters themselves. Participants that first drew with non-filtered strokes mentioned that it was difficult placing an animated stroke where they wanted it to be given its movement,

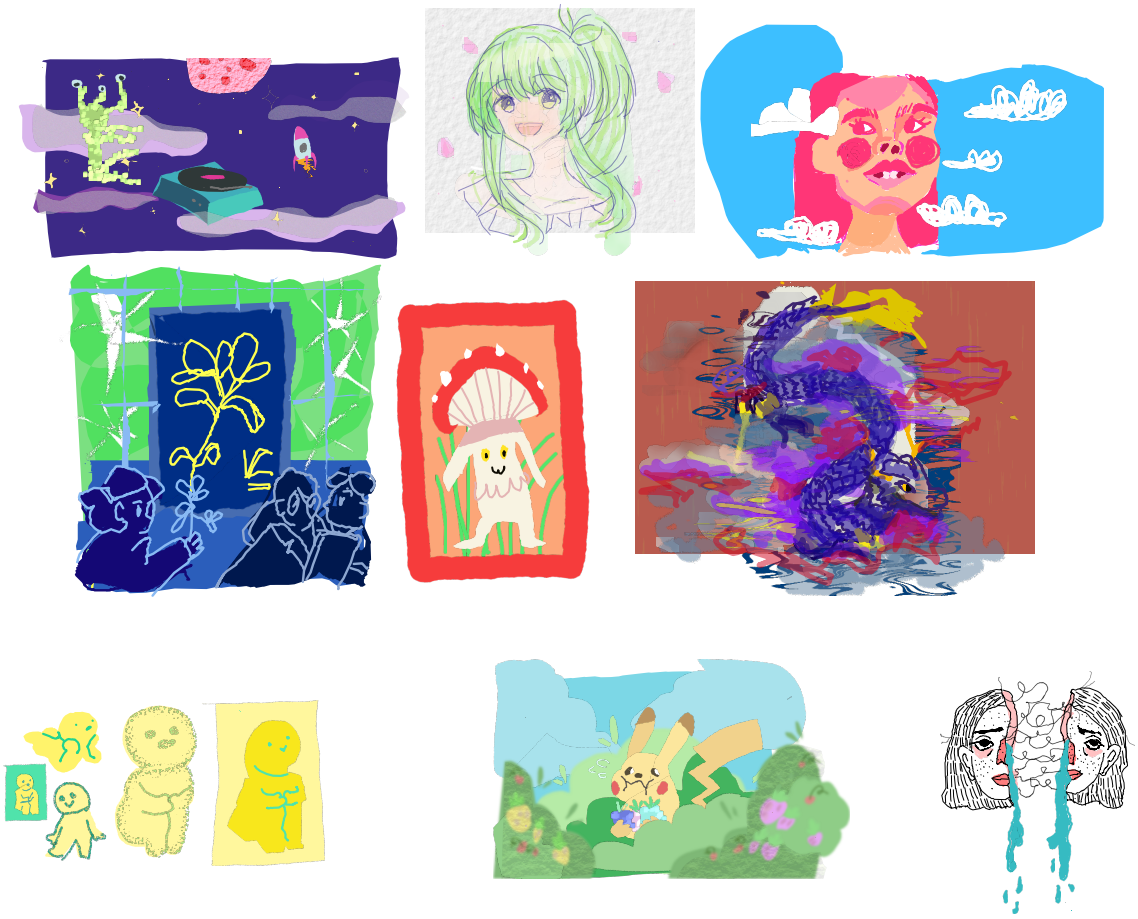

Figure 7: Final user recreations using filtered.ink

"so it was better to draw first and then click and choose what part I wanted to be animated" (P9). Participants that drew with filters directly stated that they went in with the framework of existing drawing platforms in mind, selecting and customizing a brush to draw with it directly.

Some participants also expressed concern about whether the platforms, browsers, and mediums that they publish work on would be compatible with SVGs in general. Many online publications that they illustrate for only accept standard raster image files like PNGs and JPGs, and they wished SVGs were more universally used.

5.1.3 Interface and User Experience. Despite the abstraction, some participants still noted the large *learning curve* of the filter editor, commenting that creating a filter from scratch would be intimidating to some degree. Participants particularly struggled with the Turbulence primitive, which was part of all the presets the participants chose the replicate. They relied on trial and error to understand how to correctly connect Turbulence to the rest of their primitive graph, which required the addition of another node that could receive Turbulence as an input. Participants also ran into difficulty understanding the technical names of certain primitives, such as ConvolveMatrix and Convolutions. Some participants pressed "Show Documentation" to learn more about the functions of each primitive, but only two participants with computer science backgrounds took time to skim the information, the other participants immediately exited the documentation after seeing its technical nature. There was also a notable reliance and preference for remixing existing filters among most participants. P3 stated "I don't know what a lot of the primitives mean, so the presets were easier to use and alter them, rather than start from zero." The provided presets on filtered.ink were commonly voiced to be something integral to the platform, both as a list of examples to demonstrate what the platform is capable of and as a palette of starting points for users to customize to their own needs without having to make something from scratch. P2 stated that in her own artistic practice on existing drawing platforms, "I always like to use preset brushes. So much time and thought goes into making preset brushes." However, when comparing against similar existing systems, P7 stated that "the learning curve of this is much lower than other vector art applications."

Since the affordances of node-based filter editing are similar to those of block-based programming, we used concepts from the

Cognitive Dimensions of Notation (CDN) [3] to assess usability. Using this framework can tease out nuanced design trade-offs and help distinguish between different layers of an interactive notational system [13]. Specifically, we asked the participants targeted questions on 4 CDN dimensions: diffuseness (the amount of effort required to express a desired meaning), role-expressiveness (the explainability of individual components), viscosity (the amount of effort required to make a change), and visibility (the ease with which components can be identified). On a Likert scale from 1 to 5, participants rated the tool an average of 3.1 for diffuseness, 3.7 for role-expressiveness, 3.7 for viscosity, and 3.9 for visibility. One participant specifically pointed out that because filter primitives can be connected in different ways to create widely disparate effects, "the learning curve to figure out what each of these filters do is hard." They emphasized the importance of editing on top of presets to draw semantic connections between primitive combinations and visual outcomes.

All participants mentioned the novelty and *ease of experimentation and setup* of having such an easily accessible animation tool. P6 mentioned that "I'm into making digital assets, and there's a big demand for vector animations in the industry. This platform would be good for that, it's more intuitive to use initially. If you're an illustrator without background in animation for web design, this is really helpful. There's no added burden of having to learn how to code." Some participants pointed out that they lacked prior experience in animation due to the inaccessibility and big learning curve of existing animation tools, and that filtered.ink worked as a great gateway and introduction into creating simple animations. P5 stated that "it's easier than key-framing and adding a Gaussian Blur and exporting it to a GIF. I appreciate it's scalable." Animation was noted by all participants to be one of the greatest strengths of filtered.ink and what set it apart from all existing drawing applications.

5.2 Final Filter Distribution Effects on Aesthetic Style

The participants' final drawings can be seen in Figure 7. At the end of the drawing session, all participants stated that they felt the time allocation was appropriate and allowed them to finish their illustrations. A finer breakdown of the filter distributions per drawing can be seen in Figure 8.

We decomposed each participant drawing into strokes and mapped their distributions by filter type in Figure 8 Left. All 9 participants naturally incorporated *animated* filters into their drawings, while only 2 participants incorporated *static* filters. This observation is reiterated by P4, "Animations are the coolest part, because the static effects can be achieved using existing brushes on other applications." Since the participants are familiar with raster-based tools for creating visual effects and patterns in drawing, the ability to animate in-situ was more important and useful.

Drawings with more than 200 strokes have a smaller percentage of strokes filtered ($N = 5, \bar{x} = 14.4\%, sd = 5.5\%$) in comparison to drawings with less than 200 strokes ($N = 4, \bar{x} = 59.3\%, sd = 29.7\%$). In the former more complex drawings, animated filters are used sparingly to highlight particular facial features (hair in P2, blush in P3, tears in P9) or to bring environmental elements to life (flower petals in P2, clouds in P3, rain and fog in P6). This style resembles that of the cinemagraphs in Cliplets [19] and the kinetic textures of Draco [21]. Conversely, in the latter drawings with less strokes, the filters are evenly distributed across different objects in the scene and used to convey the characters of these objects. For example, in P1's

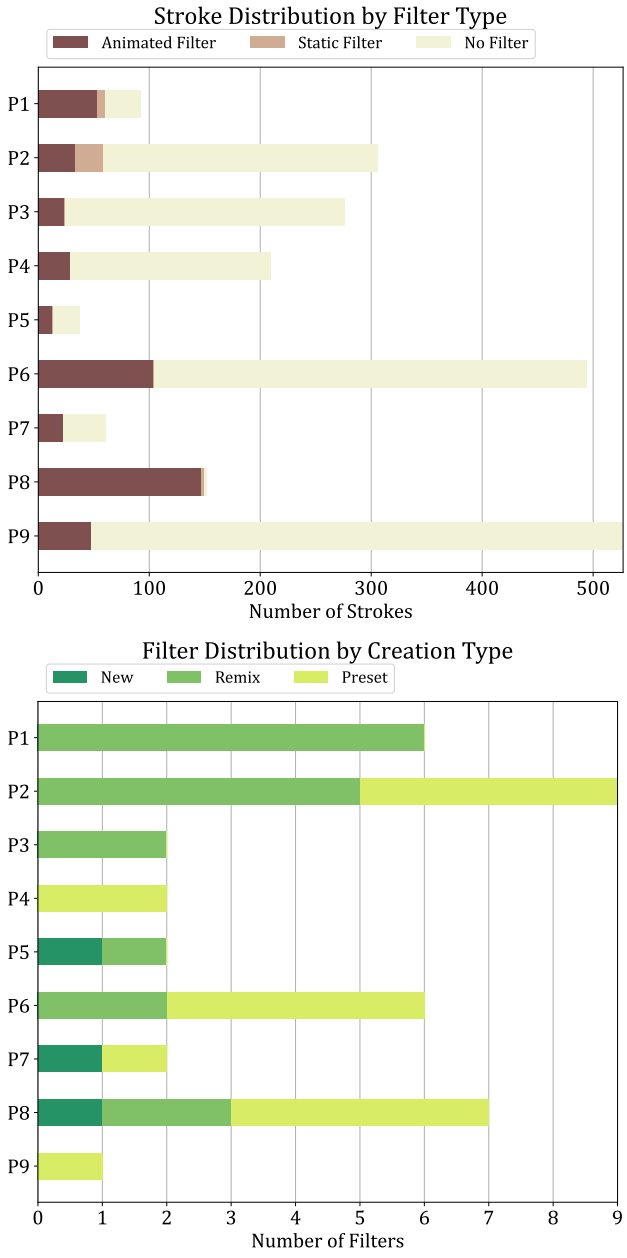

Figure 8: Left: A distribution of strokes in each participant's drawing, separated by filter type. A higher percentage of strokes are filtered in drawings with less than 200 strokes total in comparison to drawings with more than 200 strokes total. Right: A distribution of filters in each participant's drawing, separated by how the filter was created.

illustration, the entire body of the alien receives a stringy, pulsating filter to allude to its strangeness, the moon receives a rough filter to mimic its porous texture, the clouds receive a static-y filter that slowly moves up and down to evoke the feeling of floating, and the fire of the rocket receives a wispy filter that resembles smoke trails. This style is more reminiscent of the scenes from "Loving Vincent," which uses a laborious paint-on-glass animation technique [44].

Figure 8 Right contains a breakdown of the filters in each drawing by how they were created. Two participants (P4, P9) only used preset filters while two other participants (P1, P3) relied on filters they remixed from the presets. The rest of the participants employed a mixture of new filters they crafted from scratch, remixed filters, and preset filters. Given the time allocation of 30-35 minutes, participants who designed their own filters from scratch were only able to create at most one new filter for their drawing.

6 DISCUSSION

Despite limited experience working with vector graphics and no knowledge of SVG filters prior to using the system, all of the participating artists were able to incorporate filters to enhance some aspect of their illustration. Seven of the nine participants, when given the opportunity in the last free form activity to draw and edit filters freely, either remixed or created new filters that they liked enough to incorporate into their final drawing compositions. At the conclusion of the session, all of the participants were also able to correctly articulate what an SVG filter can do.

Overall, the participants were satisfied with their drawing outcomes and enjoyed the authoring experience. P8 attributed their enjoyment to "the fact that it [the strokes] could move on its own is so cute. It looks like it's living. Manipulating light was fun." Others participants also described the authoring style of the tool as "nostalgic" (P3) and "you feel like you're painting" (P1), which they noted was rare in vector-based software. By combining painterly aesthetics with easy dynamic movement, participants envisioned the tool to support concept art creation and facilitate storytelling (P2).

filtered.ink focuses specifically on using SVG filters to facilitate animated vector painting and is browser-based to support different devices and operating systems. However, our observations and findings about user workflow, drawing composition, and style can be extended to inform vector-based systems for art and design in general.

6.1 Remixing exemplars reduces the learning curve of complex operations to achieve personally preferred results.

Originally, remix was included to encourage asynchronous collaboration between users. This goal was supported to some extent, as three participants who liked the availability of preset filters created by others also uploaded their own remixed filters as new presets for others to use. However, all participants commented that the main benefit of the remix functionality is to help them understand which combinations of filter primitives can enable certain respective effects instead. By starting with an existing node-graph and tweaking internal parameters, they were able to use trial and error to parse through differences in parameter values and establish mental models of what each primitive can do.

This finding is consistent with prior work that reveals how exposure to exemplars improve creative work [24, 42] and how people can learn complex software through trial-and-error [30]. Remix provides a way to synthesize the themes of both by allowing the user to compare outcomes of trial-and-error against the original exemplar. Since there is no strict "error" in the trial-and-error process of parameter-based creative authoring, only subjective notions of more "attractive" or "favorable" outcomes to denote success cases, the ability to save instances of remix can not only help the user learn the technical details involved, but also learn specific operations necessary to achieve their *personal* desired results.

6.2 Traditional artists prefer static brushes and animated post-processing over animated brushes.

During the free-form activity where participants could voluntarily work on drawing or filter creation/editing, we observed that most people who returned to filter manipulation did so in bursts (P1, P2, P3, P6, P7, P8). That is, instead of sequentially alternating between drawing and filter-editing to see how small parameter value changes are reflected on the overall canvas, participants preferred continuous editing to create a desired filter, after which they directly apply the filter to a stroke and do not modify it again. When applying the filters to brushes, most participants (P1, P2, P3, P4, P5, P8, P9) preferred the traditional approach of drawing in plain SVG strokes, after which they used the "Change Filter" option to post-process them into animated or textured ones. When prompted, participants attributed the reason to familiarity with their original workflow. P1 stated, "Filters would be a secondary add-on because the artistic concept comes first," implying the filters are not a natural part of this pre-conceived concept. Participants also mentioned being unaccustomed to drawing with moving images and sometimes finding it distracting, with P4 stating that "it's clearer to draw a shape down and apply a filter to it. [Animated] filters are a bit difficult to see the edges of."

Two participants (P6, P7) preferred drawing directly with animated brushes instead. When prompted, P6 explained that "it's difficult to visualize what the final result will look like without drawing directly with the filter. Especially with the water filter, it looks so different from its original stroke underneath." P7 attributed their inclination towards drawing with brushes directly to the influence of their usual workflow in existing digital illustration platforms, where when they select a brush and draw with it, the brush's resulting stroke is the final result. In contrast, participants who preferred using animation during post-processing are more familiar with traditional mediums such as print (P9), where what they draw is usually static.

6.3 Animation count has an inverse relationship with illustration complexity, leading to different possible visual styles.

Although prior work such as Draco [21] and Kitty [20] have enabled dynamic textures and animations for illustration, patterns in the elements that users choose to animate are yet to be thoroughly explored. We found that participants who used our system selected different objects in the scene to animate based on the complexity of their drawing, which we equated to the number of strokes, or polylines. More complex drawings were associated with more purposeful animations applied to emphasize one or two objects in the drawing the participants wanted to stand out. This created an illusion of foreground objects moving in a loop juxtaposed against static backgrounds or moving background objects against a static foreground, a style used in cinemagraphs that can afford "monstrative pleasure transcending any narrative quality of the image" [5] and more deeply attract the viewer to the object of emphasis.

Conversely, in instances where the drawing was less complex, participants applied animations more liberally to a large majority of objects, both foreground and background. This choice resulted in a different style of illustration where each object had its own "character" that is afforded by the animation. Elements with the same filter animations applied meld into the same cohesive unit and adopt this same "character." Applying a different filter animation to each object in the scene creates a style that resembles traditional paint-on-glass animation, where each element in the scene is repainted by hand and thus occupies a slightly different position on the canvas; motion is slightly inconsistent across all elements in the drawing, creating a more organic artisanal effect.

6.4 There is a demonstrated need for consistent animated SVG support.

There is a lack of editing support for animated SVGs. Although we focused on SMIL animation specifically in our system over other conventional methods such as Javascript or CSS to preserve non-dependency, these SVG animation frameworks are not recognized and parsed in commonly used vector-based illustration and design tools such as Illustrator and Figma, nor are they preserved upon export. The same applies for SVG filters in general, although Inkscape does allow for filter import and participants have expressed comparatively less interest in this functionality. This incompatibility across systems frustrated participants; while they could import static SVGs from other vector-based applications into filtered.ink to add both static and animated effects, they could not edit these effects in the original more familiar software without losing the filter animations.

This incompatibility extends beyond SVG editing into rendering, especially on non-design or illustration focused platforms that mainly accept traditional raster formats. While a standalone animated SVG file can be opened and loaded in modern web browsers such as Chrome, Safari, and Firefox, as well as rendered within SVG or Object tags, it cannot be uploaded and previewed on social networks and other image-sharing sites. This lack of widespread systematic support deters both system designers and artists. One participant (P8) commented on this, "I can't send animated SVGs to people and post it on Instagram; I'd have to screen-record in order to show people what I made."

7 LIMITATIONS AND FUTURE WORK

Since our participants only used the tool during an 80 minute session, a third of which was spent familiarizing themselves with the application and learning about filters, future studies over a longer period of time can better reveal the creative possibilities enabled by this framework. Our system is also limited by the simplicity of its canvas interface. Participants felt that the lack of layers and limited variety of drawing tools like shapes or lasso prevented them from creating more complex illustrations. Incorporating our filter editor as a plugin for existing drawing interfaces with more comprehensive toolkits can enhance the overall drawing experience.

8 CONCLUSION

filtered.ink attempts to expand the boundaries of vector illustration by empowering artists to design lightweight textures and animations that can be applied to brush strokes via SVG filters. To accommodate the artist's visual mental model, it relies on a live node-based editor that decomposes SVG filter code into semantic components and enables users to bypass coding pre-requisites. Using stateful looping through filter parameter values, our system also allows illustrators without animation backgrounds to create fast animations. A usability evaluation of filtered.ink with hobbyist and professional artists further revealed that users gravitated towards remixing filters to learn how to create brushes that fit their personal tastes, preferred using animation for post-processing and used it differently based on the complexity of their illustration. Feedback from our participants has also indicated a need for more consistent and robust animated SVG support, both on the web and in vector-based tools. By introducing the possibility of authoring painterly vector art where the procedural brushes can be freely shared and recombined, users are afforded a new way to illustrate on the web.

REFERENCES

[1] Vladimir Agafonkin. 2020. Simplify.js. https://github.com/mourner/simplify-js.

[2] Mark D. Benjamin, Stephen DiVerdi, and Adam Finkelstein. 2014. Painting with Triangles. In *Proceedings of the Workshop on Non-Photorealistic Animation and Rendering*. Association for Computing Machinery, New York, NY, USA, 13–20. https://doi.org/10.1145/2630397.2630399

[3] Alan F Blackwell, Carol Britton, Anna Cox, Thomas RG Green, Corin Gurr, Gada Kadoda, Maria S Kutar, Martin Loomes, Chrystopher L Nehaniv, Marian Petre, et al. 2001. Cognitive Dimensions of Notations: Design Tools for Cognitive Technology. In *International Conference on Cognitive Technology*. Springer, Springer Berlin Heidelberg, Berlin, Heidelberg, 325–341.

[4] Alexandre Carlier, Martin Danelljan, Alexandre Alahi, and Radu Timofte. 2020. DeepSVG: A Hierarchical Generative Network for Vector Graphics Animation. arXiv:2007.11301 [cs.CV]

[5] Alessandra Chiarini. 2016. The Multiplicity of the Loop: The Dialectics of Stillness and Movement in the Cinemagraph. *The Multiplicity of the Loop: The Dialectics of Stillness and Movement in the Cinemagraph* 1, 1 (2016), 87–92.

[6] Ayan Das, Yongxin Yang, Timothy M. Hospedales, Tao Xiang, and Yi-Zhe Song. 2021. Cloud2Curve: Generation and Vectorization of Parametric Sketches. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Nashville, TN, USA, 7088–7097. https://doi.org/10.1109/CVPR46437.2021.00701

[7] Stephen DiVerdi, Aravind Krishnaswamy, Radomir Mäch, and Daichi Ito. 2012. Painting with polygons: A procedural watercolor engine. *IEEE transactions on visualization and computer graphics* 19, 5 (2012), 723–735.

[8] Jakub Fišer, Michal Lukáč, Ondřej Jamriška, Martin Čadík, Yotam Gingold, Paul Asente, and Daniel Sýkora. 2014. Color Me Noisy: Example-based Rendering of Hand-colored Animations with Temporal Noise Control. *Computers Graphics Forum (EGSR 2014)* 33, 4 (2014), 1–10. https://doi.org/10.1111/cgf.12407

[9] Neil Fraser. 2015. Ten Things We've Learned from Blockly. In *Proceedings of the 2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond) (BLOCKS AND BEYOND '15)*. IEEE Computer Society, USA, 49–50. https://doi.org/10.1109/BLOCKS.2015.7369000

[10] David J Gunkel. 2016. *Of remixology: Ethics and aesthetics after remix*. MIT Press, Cambridge, Massachusetts.

[11] Brian Hempel and Ravi Chugh. 2016. Semi-Automated SVG Programming via Direct Manipulation. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology* (Tokyo, Japan) *(UIST '16)*. Association for Computing Machinery, New York, NY, USA, 379–390. https://doi.org/10.1145/2984511.2984575

[12] Brian Hempel, Justin Lubin, and Ravi Chugh. 2019. Sketch-n-Sketch: Output-Directed Programming for SVG. In *Proceeedings of the 32nd Annual ACM Symposium on User Interface Software and Technology* (New Orleans, LA, USA) *(UIST '19)*. Association for Computing Machinery, New York, NY, USA, 281–292. https://doi.org/10.1145/3332165.3347925

[13] Robert Holwerda and Felienne Hermans. 2018. A Usability Analysis of Blocks-based Programming Editors using Cognitive Dimensions. In *2018 IEEE symposium on visual languages and human-centric computing (VL/HCC)*. IEEE, IEEE, Lisbon, Portugal, 217–225.

[14] Yiwei Hu, Julie Dorsey, and Holly Rushmeier. 2019. A Novel Framework for Inverse Procedural Texture Modeling. *ACM Trans. Graph.* 38, 6, Article 186 (nov 2019), 14 pages. https://doi.org/10.1145/3355089.3356516

[15] Yiwei Hu, Chengan He, Valentin Deschaintre, Julie Dorsey, and Holly Rushmeier. 2022. An Inverse Procedural Modeling Pipeline for SVBRDF Maps. *ACM Trans. Graph.* 41, 2, Article 18 (jan 2022), 17 pages. https://doi.org/10.1145/3502431

[16] Adobe Inc. 2019. Illustrator. https://adobe.com/products/illustrator

[17] Jennifer Jacobs, Joel Brandt, Radomír Mech, and Mitchel Resnick. 2018. Extending manual drawing practices with artist-centric programming tools. In *Proceedings of the 2018 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–13. https://doi.org/10.1145/3173574.3174164

[18] Jennifer Jacobs, Sumit Gogia, Radomír Mundefinedch, and Joel R. Brandt. 2017. Supporting Expressive Procedural Art Creation through Direct Manipulation. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems* (Denver, Colorado, USA) *(CHI '17)*. Association for Computing Machinery, New York, NY, USA, 6330–6341. https://doi.org/10.1145/3025453.3025927

[19] Neel Joshi, Sisil Mehta, Steven Drucker, Eric Stollnitz, Hugues Hoppe, Matt Uyttendaele, and Michael Cohen. 2012. Cliplets: juxtaposing still and dynamic imagery. In *Proceedings of the 25th annual ACM symposium on User interface software and technology*. Association for Computing Machinery, New York, NY, USA, 251–260.

[20] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, and George Fitzmaurice. 2014. Kitty: sketching dynamic and interactive illustrations. In *Proceedings of the 27th annual ACM symposium on User interface software and technology*. Association for Computing Machinery, New York, NY, USA, 395–405. https://doi.org/10.1145/2642918.2647375

[21] Rubaiat Habib Kazi, Fanny Chevalier, Tovi Grossman, Shengdong Zhao, and George Fitzmaurice. 2014. Draco: bringing life to illustrations with kinetic textures. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 351–360.

[22] Rubaiat Habib Kazi, Tovi Grossman, Nobuyuki Umetani, and George Fitzmaurice. 2016. SKUID: Sketching Dynamic Drawings Using the Principles of 2D Animation. In *Proceedings of the 4th International Conference on Mobile and Ubiquitous Multimedia* (Christchurch, New Zealand) *(MUM '05)*. Association for Computing Machinery, New York, NY, USA, 69–77. https://doi.org/10.1145/2897839.2927410

[23] Michele Knobel and Colin Lankshear. 2008. Remix: The art and craft of endless hybridization. *Journal of adolescent & adult literacy* 52, 1 (2008), 22–33.

[24] Chinmay Kulkarni, Steven P. Dow, and Scott R. Klemmer. 2014. Early and Repeated Exposure to Examples Improves Creative Work. In *Design Thinking Research: Building Innovation Eco-Systems*, Larry Leifer, Hasso Plattner, and Christoph Meinel (Eds.). Springer International Publishing, Cham, 49–62. https://doi.org/10.1007/978-3-319-01303-9_4

[25] Sang Won Lee, Yujin Zhang, Isabelle Wong, Yiwei Yang, Stephanie D O'Keefe, and Walter S Lasecki. 2017. SketchExpress: Remixing Animations for More Effective Crowd-Powered Prototyping of Interactive Interfaces. In *Proceedings of the 30th Annual ACM Symposium on User Interface Software and Technology*. Association for Computing Machinery, New York, NY, USA, 817–828. https://doi.org/10.1145/3126594.3126595

[26] Jingyi Li, Joel Brandt, Radomír Mech, Maneesh Agrawala, and Jennifer Jacobs. 2020. Supporting Visual Artists in Programming through Direct Inspection and Control of Program Execution. In *Proceedings of the 2020 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–12. https://doi.org/10.1145/3313831.3376765

[27] Jingyi Li, Sonia Hashim, and Jennifer Jacobs. 2021. What We Can Learn From Visual Artists About Software Development. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–14. https://doi.org/10.1145/3411764.3445682

[28] Yingjia Li, Xiao Zhai, Fei Hou, Yawen Liu, Aimin Hao, and Hong Qin. 2019. Vectorized painting with temporal diffusion curves. *IEEE Transactions on Visualization and Computer Graphics* 27, 1 (2019), 228–240.

[29] Raphael Gontijo Lopes, David Ha, Douglas Eck, and Jonathon Shlens. 2019. A Learned Representation for Scalable Vector Graphics. In *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*. IEEE, Seoul, South Korea, 7930–7939. https://doi.org/10.1109/ICCV.2019.00802

[30] Damien Masson, Jo Vermeulen, George Fitzmaurice, and Justin Matejka. 2022. Supercharging Trial-and-Error for Learning Complex Software Applications. In *CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 381, 13 pages. https://doi.org/10.1145/3491102.3501895

[31] Mozilla. 2021. <filter>. https://developer.mozilla.org/en-US/docs/Web/SVG/Element/filter.

[32] Srishti Palani, David Ledo, George Fitzmaurice, and Fraser Anderson. 2022. "I Don't Want to Feel like I'm Working in a 1960s Factory": The Practitioner Perspective on Creativity Support Tool Adoption. In *CHI Conference on Human Factors in Computing Systems* (New Orleans, LA, USA) *(CHI '22)*. Association for Computing Machinery, New York, NY, USA, Article 379, 18 pages. https://doi.org/10.1145/3491102.3501933

[33] Amal Dev Parakkat, Marie-Paule R Cani, and Karan Singh. 2021. Color by numbers: Interactive structuring and vectorization of sketch imagery. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–11. https://doi.org/10.1145/3411764.3445215

[34] Pradyumna Reddy, Michael Gharbi, Michal Lukac, and Niloy J. Mitra. 2021. Im2Vec: Synthesizing Vector Graphics without Vector Supervision. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. IEEE, Los Alamitos, CA, USA, 7342–7351. https://doi.org/10.1109/CVPR46437.2021.00726

[35] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, and Yasmin Kafai. 2009. Scratch: Programming for All. *Commun. ACM* 52, 11 (nov 2009), 60–67. https://doi.org/10.1145/1592761.1592779

[36] Mitchel Resnick, Brad Myers, Kumiyo Nakakoji, Ben Shneiderman, Randy Pausch, Ted Selker, and Mike Eisenberg. 2005. Design Principles for Tools to Support Creative Thinking. https://doi.org/10.1184/R1/6621917.v1

[37] John Rieman. 1996. A Field Study of Exploratory Learning Strategies. *ACM Trans. Comput.-Hum. Interact.* 3, 3 (sep 1996), 189–218. https://doi.org/10.1145/234526.234527

[38] Vidya Setlur, Yingqing Xu, Xuejin Chen, and Bruce Gooch. 2005. Retargeting vector animation for small displays. In *Proceedings of the 4th international conference on mobile and ubiquitous multimedia*. Association for Computing Machinery, New York, NY, USA, 69–77. https://doi.org/10.1145/1149488.1149500

[39] Liang Shi, Beichen Li, Miloš Hašan, Kalyan Sunkavalli, Tamy Boubekeur, Radomir Mech, and Wojciech Matusik. 2020. MATch: Differentiable Material Graphs for Procedural Material Capture. *ACM Trans. Graph.* 39, 6 (Dec. 2020), 1–15.

[40] Yang Shi, Zhaorui Li, Lingfei Xu, and Nan Cao. 2021. Understanding the Design Space for Animated Narratives Applied to Illustrations. In *Extended Abstracts of the 2021 CHI Conference on Human Factors in Computing Systems*. Association for Computing Machinery, New York, NY, USA, 1–6. https://doi.org/10.1145/3411763.3451840

[41] Ben Shneiderman. 2007. Creativity Support Tools: Accelerating Discovery and Innovation. *Commun. ACM* 50, 12 (dec 2007), 20–32. https://doi.org/10.1145/1323688.1323689

[42] Pao Siangliulue, Joel Chan, Krzysztof Z. Gajos, and Steven P. Dow. 2015. Providing Timely Examples Improves the Quantity and Quality of Generated Ideas. In *Proceedings of the 2015 ACM SIGCHI Conference on Creativity and Cognition* (Glasgow, United Kingdom) *(C&C '15)*. Association for Computing Machinery, New York, NY, USA, 83–92. https://doi.org/10.1145/2757226.2757230

[43] The Inkscape Project. 2020. Inkscape. https://inkscape.org

[44] Tom Van Laerhoven, Fabian Di Fiore, William Van Haevre, and Frank Van Reeth. 2011. Paint-on-glass animation: the fellowship of digital paint and artisanal control. *Computer Animation and Virtual Worlds* 22, 2-3 (2011), 325–332. https://doi.org/10.1002/cav.406 arXiv:https://onlinelibrary.wiley.com/doi/pdf/10.1002/cav.406

[45] David Weintrop and Uri Wilensky. 2017. Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Trans. Comput. Educ.* 18, 1, Article 3 (oct 2017), 25 pages. https://doi.org/10.1145/3089799

[46] Zhengxia Zou, Tianyang Shi, Shuang Qiu, Yi Yuan, and Zhenwei Shi. 2021. Stylized Neural Painting. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, Nashville, TN, USA, 15689–15698. https://doi.org/10.1109/CVPR46437.2021.01543