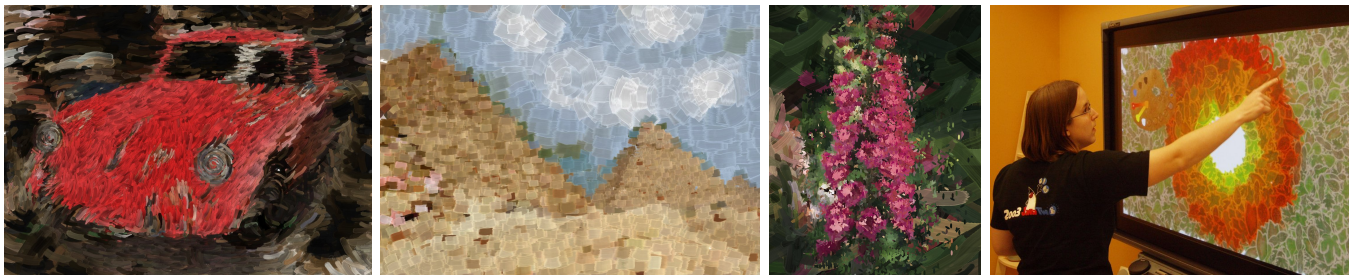
Modeling with Rendering Primitives: An Interactive Non-Photorealistic Canvas

Martin Schwarz* Tobias Isenberg* Katherine Mason* Sheelagh Carpendale*
Department of Computer Science, University of Calgary



Abstract

Non-photorealistic rendering has placed much emphasis on developing algorithms that determine the appearance of renditions. To successfully deploy NPR rendering systems using these algorithms, however, one has to consider how artists, illustrators, or lay people can influence the created renditions. Many systems require a cyclical process of parameter tweaking, rendering, and validation before one is satisfied with the final rendition. We present an interactive NPR canvas in which a user can construct a rendition with pre-rendered primitives and modify these primitives using tools that provide spatially explicit computational assistance. We call this approach *modeling with rendering primitives*. Our technique has the advantage of algorithmic support for creating NPR renditions but requires neither global parameter adjustments and re-rendering cycles nor attribute changes on individually selected primitives. We demonstrate the applicability of this interaction technique for the creation of painterly rendering, pointillism, and decorative mosaics.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.3.m [Computer Graphics]: Miscellaneous—Non-Photorealistic Rendering

Keywords: Interaction techniques, non-photorealistic rendering, modeling with rendering primitives, stroke-based rendering.

1 Introduction

As the field of computer graphics matures, the methods to create rich and subtle digital visuals continue to become more sophisticated. These results range from increasingly realistic to a variety of more expressive and abstract styles. Underlying this range of results are two basic types of representations: vector-based or pixel-based. A vector-based representation fairly readily affords subsequent adjustment but is somewhat limited in its ability to represent visual richness due to its algorithmic character. A pixel-based representation, in contrast, can be adjusted on a large scale (e. g., globally using image filters) or on a very small scale (i. e., per one or a few pixels). Making changes to aspects of the pixel image in between these two extremes, however, is much more challenging and raises many issues, including the selection of appropriate aspects or regions that will make the desired change possible. Still, pixel images afford a richness of visuals which makes them a desirable choice. It is the problem of providing interactive freedom while retaining as much visual richness as possible that we address.

While one could argue that any amount of richness is available if one is prepared to place and color individual pixels, realistically this interaction needs to happen on a more meaningful level. One example is the use of common artistic primitives, such as brush strokes, that represent elements of the depicted scene (e. g., leaves of a tree or waves on water). This visual richness is apparent in non-photorealistic rendering [Gooch and Gooch 2001; Strothotte and Schlechtweg 2002], where many techniques simulate traditional media, artistic techniques, and illustrative styles, producing astounding results. NPR research, however, tends to focus on the algorithms needed to achieve a chosen style. Algorithms are commonly presented as a 'black box' to users of a system, allowing interaction only through parameters, after which the rendering produces a result. This common lack of interactive intervention during the rendering process may be part of the reason why NPR methods have had only limited adoption within the artistic community. *"In my experience with non-photorealistic rendering, I am often frustrated by my inability to stop, reach into and tweak an automatic process. To make painterly rendering techniques more useful for production-quality work, we need to develop algorithms and interfaces that get the artist in the loop"* [Seims 1999].

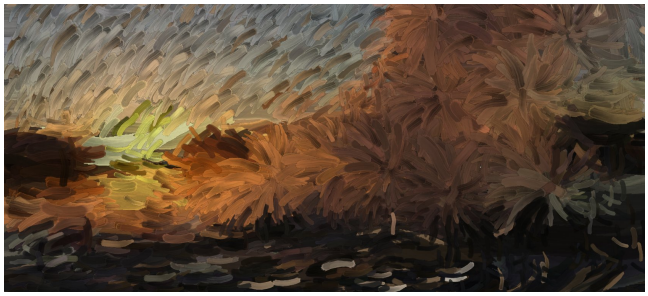
In this work, we focus on the interaction rather than developing simulations of traditional techniques. We contribute a new approach for interactive image creation in which an image is constructed with pre-rendered primitives through an interactive process that provides spatially explicit computational assistance. We think of this process of image construction as *modeling with rendering primitives*. We demonstrate our approach using an interactive environment for working with non-photorealistic rendering styles, such as painterly rendering, pointillism, and decorative mosaics. The primitives can be modified after having been created instead of being permanently placed and parameterized.

We offer a range of methods that support interaction during the rendering process enabling a wide variety of ways to create images. Figure 1 shows an example, illustrating a possible use of our system.

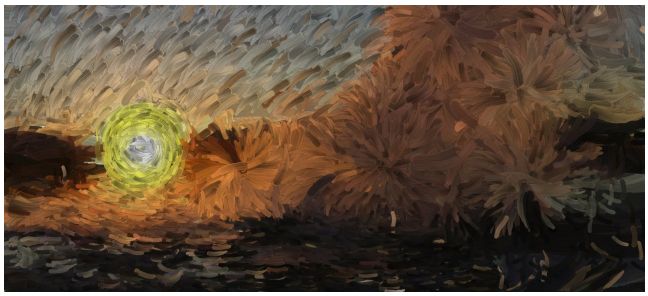
*e-mail: {maschwar | isenberg | katherim | sheelagh}@cpsc.ucalgary.ca

(a) Source image to derive color distribution.

(b) Background layer.



(c) Middle layers added.



(d) Foreground layers added.

Figure 1: *Walking through the creation of an example image.*

First, we start by creating the background. In order to start from a meaningful color distribution we load a source image to be used as a basis (Figure 1(a)). Next, we interactively fill this first (background) layer with strokes. The created strokes are relatively large and give the general color impression as well as direction of painting (Figure 1(b)). Then to provide more detail, smaller strokes are added, again using the source image to determine the color distribution. These strokes are interactively adjusted and manipulated to achieve the desired effect. For example, the strokes in the sky were given a more unified direction, the water another, and the spray was portrayed using radial orientations in several places to express its dynamic character (Figure 1(c)). Thus, in Figure 1(c)) we have a reasonable image; however, the rising sun does not seems to have the right visual impact. Since all strokes are still active, further adjustments to quantity, size, color, and orientation can be done until the desired result is achieved (Figure 1(d)).

This walk-through illustrates our contributions: all strokes remain interactive no matter what their characteristics or when they were applied, any parameters can be adjusted interactively with immediate visual response, and the visuals are rich rasterized textures throughout the interaction. In addition, we can use a variety of visual primitives (as many different types as desired) and new primitives can continually be added or erased as individual primitives or groups of primitives. That is, the work done on the image, while being continually visually rich, does not become permanent; all can still be interactively manipulated no matter how many primitives of however many styles for whatever length of time.

The remainder of the paper is organized as follows. In Section 2 we review previous work in the areas of digital painting, NPR techniques, and interaction with NPR. We then develop our concept for supporting modeling with rendering primitives in Section 3. Based on this concept, Section 4 introduces our system details and its implementation aspects. In Section 5 we discuss artists' reactions and show example results. Finally, in Section 6 we conclude the paper.

2 Related Work

We start by contextualizing this research within recent digital painting systems and NPR techniques such as stroke-based rendering, and then discuss closely related NPR interaction techniques such as stylistic painting, interaction with strokes, and control of spatial data structures.

Digital painting systems: These are very heavily employed today in the creation of digital art. There are two major categories of systems: pixel-based (e. g., Adobe PhotoShop and Gimp) and vector-based (e. g., Adobe Illustrator and Inkscape). Pixel-based systems allow users to draw one stroke at a time using a variety of tools and each stroke is rasterized and embedded in the canvas. In the second category, users can also draw stokes but these are stored in a parametrized (vector) form that is rasterized on-demand for display. With respect to the representation of primitives, our approach can be thought of as a *hybrid* of these two approaches. We take advantage of the visual richness provided by pixel-based systems, but also maintain all our rasterized primitives just as actively adjustable as in vector approaches. Thus, we provide good visuals while working and continued freedom for subsequent changes.

Stroke-based rendering: Our work makes use of higher-level primitives as developed in NPR research. Such primitives include brush strokes as in painterly rendering [Meier 1996; Hertzmann 1998; Park and Yoon 2004], pointillism [Yang and Yang 2006], mosaic tiles [Hausner 2001; Elber and Wolberg 2003; Di Blasi and Gallo 2005], and stipple points [Deussen et al. 2000; Secord 2002; Schlechtweg et al. 2005]. Together these are commonly referred to as stroke-based rendering [Hertzmann 2003] and most simulate traditional techniques of artistic expression or illustrative depiction. In addition, the boundaries of such traditional depiction have been pushed by, e. g., dynamic primitives such as graftals [Smith 1984; Kowalski et al. 1999; Markosian et al. 2000] which can algorithmically generate geometry in the rendering process depending on the current view and other parameters.

Triggering algorithmic response: Hertzmann and Perlin [2000] describe interaction with a NPR painting by affecting the video stream that is used as input. The automatic NPR process then paints over only those parts that have changed. Interaction on an even more global level is possible by soliciting user input for rating a number of suggestions that a genetic algorithm has produced

[Grundland et al. 2005; Collomosse 2006]. However, in these approaches the interaction is separate from the rendering process and, thus, lacks the immediate visual feedback we provide.

Interactive stroke placement: Other examples that focus on interactivity include systems that model the artistic painting process by interactively applying strokes. For example, Curtis et al. [1997] employ physical simulations to replicate the effects of watercolor painting, which can then be used interactively to composite paintings. Baxter et al. introduce an interactive painting system with haptic feedback and a deformable 3D brush model [2001] as well as a physically-based interactive model for paint [2004] that both aim at creating a painting experience that comes close to the actual painting process. In contrast to such methods that closely simulate real paint, Ryokai et al.'s I/O Brush [2004] captures real world textures for use as strokes in digital painting. Other approaches concentrate on realizing three-dimensional painting techniques [Keefe et al. 2001; Schkolne et al. 2001]. Our work is inspired by the immediate visual feedback that all these approaches provide. In contrast to such stylistic painting systems, however, we provide interaction beyond the stroke placement: our primitives do not become permanent and can be manipulated even after they have been placed.

Interaction with strokes: Some systems allow users to interact with image elements after their placement. For instance, Salisbury et al. [1994] provide interaction with strokes, allowing users to adjust the number of strokes to be used and to change their waviness. Deussen et al. [2000] use brushes to interactively reposition or resize stipples as well as create new ones or delete them. The WYSIWYG-NPR system [Kalnins et al. 2002] allows changing the rendering style by painting example strokes, which are then used as templates for algorithmically stylizing other strokes. Negotiating Gestalt [Mason et al. 2005] uses a multi-agent system to model the image creation process as coalition forming, allowing users to take direct control of agents or coalitions during the process. All these systems step away from the typical 'black box' rendering approach and their use of specialized tools to interact with image elements has informed our own work. However, most are still limited in the way they require selecting the elements with which to interact or allow interaction only indirectly through agents.

Control through spatial data structures: Most closely related to our work in terms of stroke control are Haeberli's Paint by Numbers [1990], Salisbury et al.'s orientable textures [1997], Olsen et al.'s [2005] interactive vector-fields, and the RenderBots system [Schlechtweg et al. 2005]. Paint by Numbers uses a source image and a canvas. One can brush the source image, resulting in a stroke placed on the canvas using the location and color collected from the source image. While this provides considerable creative freedom, once the strokes have been placed, they have become part of the image and are no longer accessible individually to change their properties. In a related approach, Salisbury et al. [1997] allow users to paint the orientation of hatching strokes onto a 2D image as well as to locally edit the tone. Based on these two values, strokes from a stroke textures are oriented and drawn. While producing print-quality results, the system is restricted to hatching and the two types of input. Also, Salisbury et al. separate between an interactive phase to specify the tone and orientation maps and the non-interactive rendering phase so that users cannot immediately see the results of their interactions. Olsen et al. [2005] use interaction with a separate vector-field in a similar way to control the placement of painterly brush strokes. Finally, RenderBots combine a multi-agent system with NPR rendering. Here, a user can brush autonomous agents onto the canvas, which then read values from a stack of G-buffers and change their behavior depending on these values. Even though this technique opens new avenues for rendering, it is restricted in that it only uses pre-rendered buffers for influencing the agents' actions and interaction through agents is still indirect. However, we draw from these ideas of spatial control: the source image in [Haeberli 1990] and the orientation maps in [Salisbury et al. 1997; Olsen et al. 2005] can be seen as buffers that store information about the rendering process. By making these buffers interactively adjustable at any stage we increase the control over the rendering process and add new possibilities for the interactive creation of non-photorealistic renditions.

3 Modeling with Rendering Primitives

We focus on the specific problem of providing *interactive* techniques to create, adjust and manipulate NPR renditions. Effective interaction techniques should ideally allow the artist to use their personal taste and requirements to influence a particular rendition. Also, it is essential to provide *intuitive* interactivity continual adjustments, immediate visual feedback, and temporal coherence. Providing these aspects is challenging in many NPR systems due to the nature of their rendering process, which can be roughly described as follows. First, a model is constructed or chosen. Models may include 3D geometry (e. g., CSG or 3D mesh representations), volumetric data, or digital 2D images. Then, a rendering method is chosen, usually also requiring the specification of a multitude of parameters that affect the final rendition. Both of these first steps are frequently interactive. However, after these choices are made, the rendering process is started and typically proceeds without interruption and as a consequence without interaction. The resulting rendition can be modified in a post-processing step.[1] If the results were not fully satisfying, it is possible to adjust the model and/or the chosen parameterization and to restart the rendering.

In general, interactive influence on the outcome of the rendering is only possible during modeling and parameterization. The algorithmic processing itself has been defined by a programmer and usually cannot be interrupted or changed while the model is being rendered. While some systems support interactive creation of images in which users can interactively apply brush strokes, adjustment is usually not possible once strokes have been set down. This restriction of artistic input does not support our goal of continual adjustments, immediate visual feedback, and temporal coherence of interaction and rendering. By turning the *complete rendering process*, as described above, into an interactive process with immediate feedback, the user can choose to interrupt the rendering at any point, choose new rendering styles, tweak brush strokes to influence the artistic impact, find brush stroke arrangements where algorithmic approaches are unsuccessful, or experiment with how rendering primitives are placed. This provides creative freedom at all stages of the image production process is a worthwhile goal.

In order to achieve this goal we introduce the concept of *modeling with rendering primitives*. A *rendering primitive* is defined as any small part from which images can be created, including different types of brush strokes such as lines, points, dabs, dots, mosaic tiles, etc. In its simplest computational form, a rendering primitive can be a pixel—a single cell in a rasterized digital image. In our interactive NPR canvas we use three types of rendering primitives: a set of brush strokes as can be found in current paint systems, tiles to construct decorative mosaics, and pointillist paint blobs. Modeling with these primitives is supported in that one can continuously, create, take apart, re-assemble and adjust, essentially building one's image out of a variety of rendering primitives.

Once different rendering primitives have been assigned to a spatial location on a rendering surface, they form the model to be rendered.

[1] While some NPR techniques (e. g., those that use images as input) can be seen as post-processing techniques, they can also be considered to follow the above process: starting with an image as the model, they perform a rendering process using this model to produce another image.

At this point our approach deviates from the standard iterative rendering process in that it offers interactive support beyond global parameter changes or localized interaction on pre-selected primitives. We use mobile tools that have localized effects on the NPR canvas to control the properties of rendering primitives in their regions of influence. The type and spatial location of this algorithmic support can be interactively changed *throughout* the rendering process. This means that we can *model with rendering primitives* by building and adjusting the model as the work progresses. Since the primitives are fully rendered, the image takes form through the interaction with them. This concurrent modeling and rendering is essential: the model is created and constantly updated as rendering primitives are generated, manipulated, and removed.

4 An Interactive NPR Canvas

To enable modeling with rendering primitives, our system requires: (1) a mechanism that can efficiently support simultaneous manipulation of many primitives, (2) tools to assign meaningful parameters and useful interactions, (3) an interface to coordinate the tools, and (4) efficient rendering techniques to maintain interactive frame rates, which we will address in the following.

4.1 Affecting Primitive Properties

Affecting properties of a large number of primitives while maintaining responsiveness of the system presents a challenge. As was alluded to above and as is a common solution to graphic problems, we use a stack of two-dimensional interactive buffers (*i-buffers*, as opposed to static G-buffers) as the basic structure of our system to address this issue. Each i-buffer holds information for the rendering such as color, orientation, shape, or movement. The information is placed into the i-buffers interactively through tools which locate the data spatially within the i-buffer. Therefore, a primitive can determine how to render itself by looking up the data stored in the i-buffers. This way it is possible to manipulate the properties of entire regions of primitives without having to select any of them individually. I-buffers are maintained as a separate spatial data structure that is easy to manipulate and fast to query. The respective buffers are matrices of scalar values (e. g., for the size) or vectors (e. g., for orientation and color), depending on the dimension of the represented properties. An additional benefit of using this buffer approach is that it incorporates improvements in interaction responsiveness [Isenberg et al. 2006].

In practice, there are two main categories of i-buffers that we maintain: *persistent buffers* and *instantaneous buffers*. Persistent buffers represent actual property values such as color, size, orientation, and shape. They exist for the entire run-time of the system and are constantly updated in response to user interaction. Instantaneous buffers, on the other hand, represent changes to properties that cannot be maintained in persistent buffers. Such properties include position and existence, since a primitive needs to exist at a position in order to be able to query a buffer. Information in instantaneous buffers, therefore, exists for one rendering step only and the buffer is reset afterwards. For example, we may want to move primitives in a region by a certain distance using a motion tool. A vector representing this motion could be rendered into a movement buffer and primitives use that data to offset their position. It is, however, reset after this frame's animation has been completed so that primitives do not keep moving even after the use of the motion tool is finished.

We support layering of strokes, inspired by the method artists employ to build their scenes from background to foreground. As shown in the example in Figure 1, this allows us to isolate regions on the canvas and to enable working with them independently. Layering requires that several stacks of buffers are used to control the different layers of primitives. As we need only one active buffer stack, we store the remaining inactive buffer stacks on the hard drive, saving memory. This means that primitives in inactive layers can no longer read their properties from a buffer stack. This is not necessary, however, since as long as they are inactive they are not changing their properties. Thus, inactive primitives can just use the properties from when they were last active and do not need to query any data until they become active again.

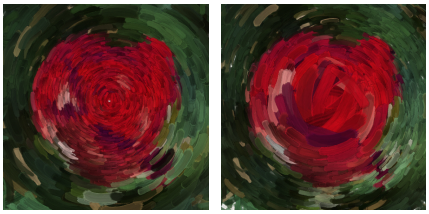
4.2 Tools

Tools are the means by which the i-buffers are manipulated. This manipulation, in turn, changes the behavior of the rendered primitives. Similar to digital painting applications, they are inspired by the brush-and-canvas metaphor from real painting. In contrast to previous approaches, however, our tools manipulate primitives indirectly by 'painting' into the above mentioned i-buffer stack. Our system provides tools (and, therefore, buffers) to manipulate the color, size, orientation, and shape (e. g., of pointillism dots) of rendering primitives as well as to create, move, and delete them. To make changes a tool renders new values into the respective i-buffer according to the area it covers on the canvas. A gradual change from the previous values on the perimeter of the tool's influence range is achieved using attenuation.

(a) Original color. (b) Small change. (c) Drastic change.

Figure 2: *Affecting the color of primitives.*

Color tool: Depending on its attenuation, the color tool mixes the new color with the previous color values read from the color buffer (Figure 2). The 3D color vector is represented using the painterly Red-Yellow-Blue (RYB) model [Gossett and Chen 2004] instead of the more common RGB model. RYB was chosen because it allows color mixing that is inspired by what is expected from mixing pigment colors. Primitives reading the RYB color from the buffer then convert it to RGB for rendering.

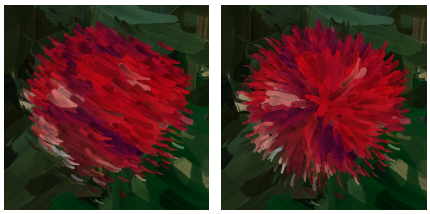


(a) Strokes smaller. (b) Strokes larger.

Figure 3: *Resizing elements.*

Size tool: Similar to color, this tool changes the size of objects by reducing or increasing the local size-buffer values (Figure 3).

Orientation and motion tools: These tools both have effects either independent from or dependent on the tool's motion while being used (Figures 4 and 5). The orientation tool lines up primitives to follow its motion path (Figure 4(a)) or imposes motion-independent orientations (Figures 4(b) to 4(d)). Likewise, the motion tool forces



(a) Following the tool. (b) Radially.

(c) Circumpolar. (d) Randomly.

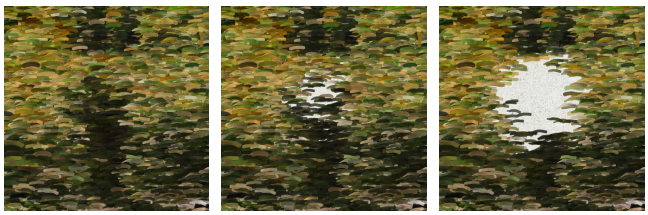
Figure 4: *Changing primitive orientations interactively.*

(a) Following the tool. (b) Attraction. (c) Repulsion.

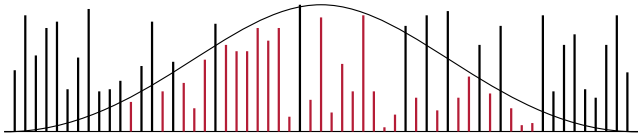
Figure 5: *Circular motion tool affects the position of primitives.*

primitives to follow the movement of the tool (Figure 5(a)) or exerts an attraction (Figure 5(b)) or repulsion (Figure 5(c)) force on primitives with respect to the tool's current position. Both are realized by rendering 2D vector fields into the respective i-buffers.

Eraser tool: This tool slowly erases objects (Figure 6), supporting a slow thinning-out of regions on the canvas. For this purpose it renders attenuated probabilities for objects to delete themselves into the erase-buffer (curve in Figure 6(d)). Each primitive in turn computes a random number in $[0, 1]$ and compares it with the read erasing probability to determine if it should delete itself (spikes in Figure 6(d), the red ones pass the test and the corresponding objects



(a) Original. (b) Few strokes erased. (c) Continued erasing.



(d) Attenuated erasing with probabilities (curve) which are compared with random numbers (spikes) computed by the objects (one row within the erasing buffer shown; red spikes represent objects to be erased).

Figure 6: *Gradual, attenuated erasing of elements.*

are deleted). This way we can specify a parameterizable deletion of primitives as well as a gradual change of the probability across the surface covered by the erasing tool.

New strokes tool: This tool differs from others in that it does not use i-buffers. Instead, it creates new primitives randomly distributed within its range, selected from a series of primitive types, as densely or sparsely as specified by the user.

4.3 Interface Elements

In addition to the tools described above, our interface consists of two more elements: (a) a canvas, where primitives are placed and which holds the property buffers and (b) a palette to control the tools, to specify what primitive properties to manipulate and with what parameters. The communication between these is realized using a parameter-buffer (Figure 7) that maintains system state and

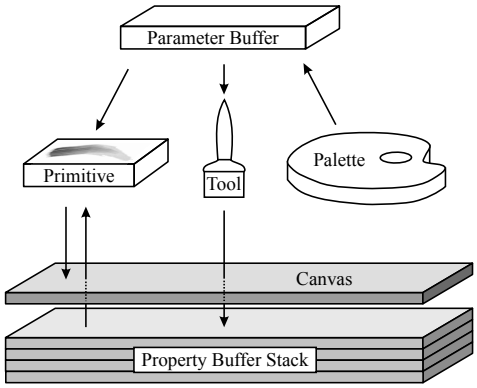
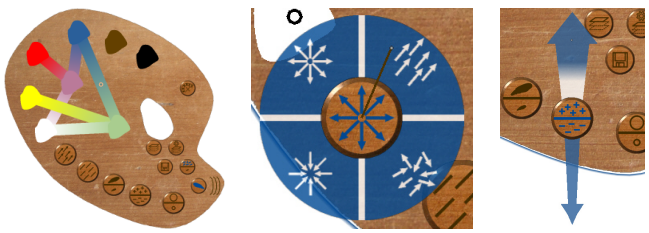


Figure 7: *Communication between interface elements.*

tool settings data. Using a parameter buffer instead of direct communication has the advantage that both tool and palette can be treated as special primitives that also use buffer access to do their communication. As the number of values needed for communication does not depend on the canvas size, the parameter buffer is the only buffer that has a fixed size, usually much smaller than the rest of the buffers. Primitives also read from the parameter buffer to know about the current layering state, causing them to either read from the regular buffer stack or to use their stored property values.

The palette as the control center (Figure 8) provides means for tool selection and parameter specification (including tool size), communicating with the tools via the parameter buffer. The palette also affords color mixing (Figure 8(a)). This process is initiated by dragging color blobs from the perimeter to the inside (inspired by [Meier et al. 2004]). Dropping one color onto a color blob already in the mixing area causes them to mix and a bar with color gradients from



(a) Color mixing. (b) Pie menu. (c) Slider.

Figure 8: *Palette as a central control.*

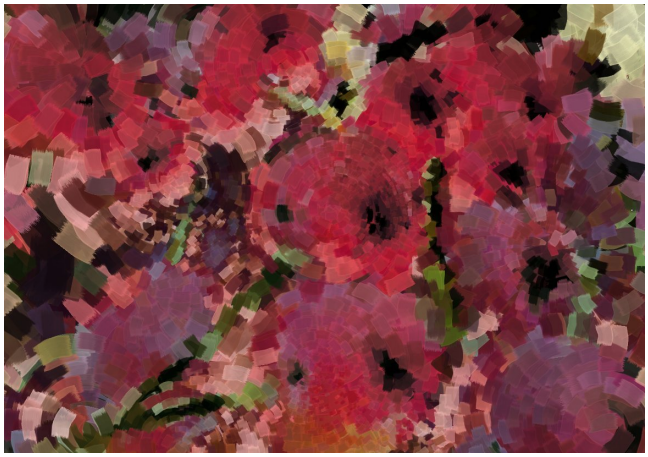


Figure 9: *Wider strokes create an abstraction effect.*

Figure 10: *Longer strokes bring out the water reflections well; pointillism was used to portray the land part.*

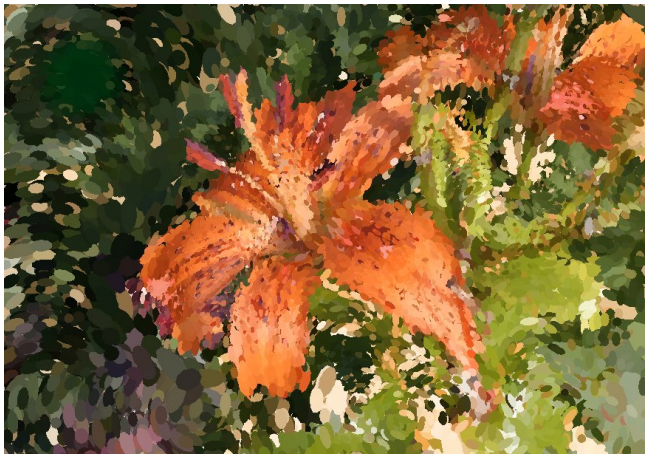


Figure 11: *Pointillism using simple geometric shapes.*

the original color to the mixed one is shown. Colors can now be chosen from any point on the color blobs or gradients. The other interface elements on the palette are represented by buttons, pie menus (Figure 8(b); [Hopkins 1991]), and sliders (Figure 8(c)) to enable click-less operation using only touch and move interactions.

4.4 Efficient Rendering and Performance

To maintain the system's responsiveness, it is essential to render efficiently. For our system this not only includes the rendering of the primitives to the screen but also the rendering of values to the property buffer stack. The former is realized through OPENGL, representing the NPR strokes as semi-transparent textures, and pointillism dots and decorative mosaics tiles as simple shapes. While it would be possible to maintain the i-buffers in graphics memory (to take advantage of hardware-accelerated rendering for this task), the necessary read-backs from graphics memory for primitives to look up i-buffers data would be expensive. Thus, we maintain the i-buffers in main memory and use pre-computed stencils of tool values to facilitate fast i-buffer updates.

In practice, our system is able to manage a large number of primitives simultaneously as well as allow interaction with them while maintaining interactive rendering rates. We tested it both in a traditional desktop setting (1,400 × 1,050 pixels; 1.47 mega pixels) as well as on a large (146 cm × 110 cm), high-resolution (2,800 × 2,100 pixels; 5.88 mega pixels) tabletop display that affords direct touch input for interacting with the system. Both settings were driven by the same 3 GHz dual core machine with 2 GB RAM and two 512 MB nVIDIA GeForce 7900 GTX. Frame rates ranged from 177 fps (1K strokes) to 13 fps (16K strokes) in the desktop setting and from 29 fps (1K strokes) to 8 fps (16K strokes) for the table, using 256^2 mip-mapped textures each covering about 128^2 pixels.

5 Using the Interactive NPR Canvas

Throughout the process of developing our system and after completion we asked our colleagues and four professional artists to evaluate it. This section reports on the results of this informal evaluation and presents additional examples created with our system.

5.1 Example Images

Figure 9 shows the use of quite broad strokes to create an abstract effect. To work out detail and to achieve a contrast to the bold strokes in the background we adjusted the size of the strokes interactively in some regions. In contrast, Figure 10 uses long and thin strokes as they capture the structure of the tiny waves on the water and, thus, the character of the reflection quite well. The strokes were arranged parallel to the water line to produce this effect. This example shows the advantage of interactive stroke manipulation—it remains difficult for most automatic painterly rendering algorithms to reliably find orientations for brush strokes in areas such as the smooth water surface or a clear sky [Park and Yoon 2004].[2] Since long strokes caused undesirable artifacts in the trees and rocks, we used pointillism strokes for this region instead, showing how to combine several primitives. Figure 11 uses only pointillism blobs. Here, orientation of the strokes in the background was adjusted such that it gives the impression of a larger green plant, maybe a bush, behind the tiger lily. Figure 12 shows the use of mosaic tiles which have been manually oriented to indicate groups of the surrounding water plants as well as features of the fish itself. Their distribution is based on forces derived from a tile position buffer. Finally, Figure 13 shows a painting created without a source image using colored leaves as

[2]Some researchers have addressed this problem of automatically deriving stroke orientations for areas with small gradient magnitudes including Litwinowicz [1997] as well as Hays and Essa [2004]. Using interaction, however, our technique offers more artistic freedom for this process.

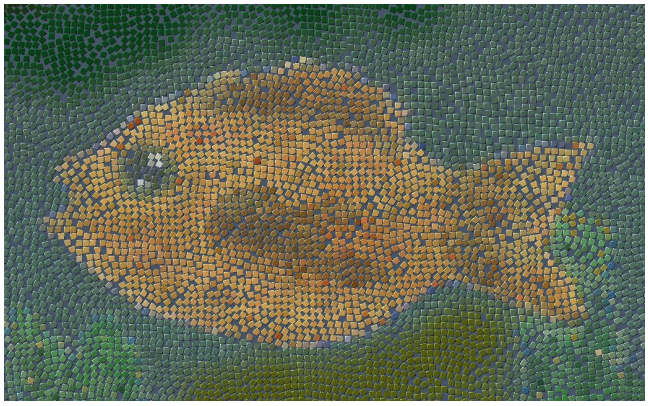


Figure 12: *Decorative mosaics, tiles with shading effects.*

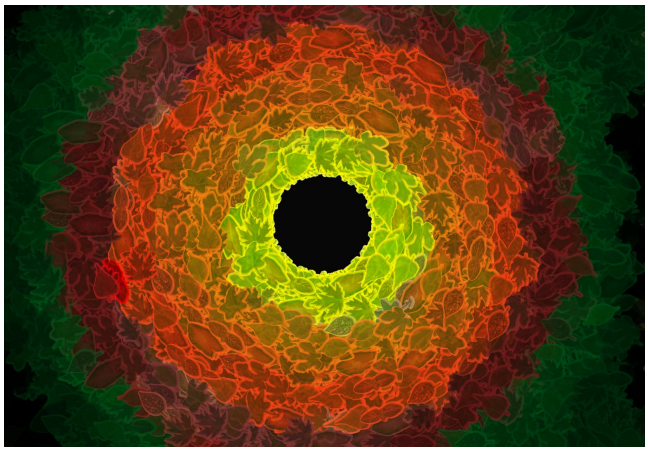


Figure 13: *Free-hand painting with leaves as primitives.*

primitives, inspired by works by Andy Goldsworthy.

5.2 Comments from Artists

Artists using the program noted that it did not feel like a paint program to them, rather it was an entirely new experience. One artist said that it felt *"like working with collage elements."* They liked the effect that new strokes being added to the canvas *"take on the range of hue of the strokes already laid down below."* Loading images into the color buffer caused *"a subtle flow of hue with lots of variation from stroke to stroke,"* due to variance present in the color distribution of the loaded image and in the textures of the strokes. Artists were intrigued by the effect that single strokes, if tossed across the canvas, would *"change hue along the way according to the landscape below. A kind of chameleon element. I liked that."* In general, artists appreciated the possibility of influencing the properties of elements on the canvas after these had been created, saying that it *"feels like an NPR filter that you can play with and change"* and that *"the interactivity of the tool made working with it much more interesting compared with traditional NPR filters."* One artist reported that *"the mobility of all the elements, as if suspended in some kind of liquid surface is a rather unique aspect [...] but it is not like the viscous surface of paint—more like a watery surface where the individual elements retain their distinct edges (unlike watercolour painting). Like life underwater ..."* However, artists also felt that the ability to draw a line is missing which will need to be added in the future. Users also missed the option to create new stroke textures themselves to be used in the program. In general, users initially had some difficulty adjusting to the different paradigm of painting. One comment was that it feels *"something like trying to compose a picture with feathers that keep moving about in the air."* However, after getting used to the new approach, people liked it and said that it was easy to create images in a very short time.

We also received feedback from the artists about the interface, leading to several changes to incorporate their input and to address their concerns. For example, the use of layers to separate regions from one another and the ability to remove layers altogether were included after comments from artists. Similarly, they suggested showing inactive layers transparently to improve the understanding of the active layer. The color mixing interface using drag-and-drop color blobs was also improved according to requests from users.

6 Conclusion and Future Work

In summary, we have explored the creation of and interaction with non-photorealistic rendering through an 'adjust while observing' approach—instead of 'tweak, render, observe and tweak again.' As our render process is instantaneous and our changes are local, we make it possible to influence the primitives as they are assembled on the canvas and to modify aspects as desired through fluid and continuous interaction rather than by indirectly adjusting parameters. *Modeling with rendering primitives* opens up possibilities for interaction with the image generation process, making it possible *"to stop, reach into and tweak [the] automatic [rendering] process"* [Seims 1999]. We have presented a system that implements this approach and have explored how it enables artists to take control of the image production process.

While our concept opens up possibilities for building tools that enable interaction directly with the rendering process, there are still some limitations. The approach concentrates on small strokes so that longer elements such as outlines are still difficult to generate. In addition, the primitives read their properties from i-buffers which leads to primitives changing as they are moved across the canvas. This is caused by the indirect interaction with primitives through buffers. A more direct way would be to include options for using only instantaneous buffers that just apply changes to the properties of elements which are otherwise maintained by the elements themselves. Also, a more diverse set of tools could be explored. This includes providing more or less algorithmic support (more support could be given by applying image processing to small or large i-buffer areas, e. g., using Gaussian blur or averaging vector field directions), representing other primitive properties (e. g., more/other parameters for shape control), and using other and/or more powerful primitives (such as graftals to increase the power of expression or primitives from pigment-based media such as watercolor). In this context, it would also be interesting to incorporate larger primitives—rigid or flexible—that have more than one point to query property buffers and which could be used to represent elements such as outlines. Similarly it would be interesting to load images into buffers other than the color buffer, including G-buffers [Schlechtweg et al. 2005] and other images.

As we developed the system to be applicable to large displays with multi-touch input, we would like to explore the use of multiple inputs in the interaction such as multi-finger painting. We are also interested in the difference between using our system in traditional desktop environments as compared to large, high-resolution, direct-interaction tabletop or wall displays. Indirect ways of interaction by showing the buffer that is being adjusted on a separate screen while the rendering adjusts on the main screen may be similarly exciting. Finally, the i-buffers are currently maintained at screen resolution. As we use interpolation to query non-grid locations, smaller buffer

sizes may be beneficial to the system's performance.

To conclude, we see our contribution as extending the types, style, and temporal availability of interaction during the rendering process, rather than in simulating a specific style. In fact, other stroke-based NPR techniques could serve as input for our system, using our interaction possibilities for providing possibilities for touch-up. This way users can take advantage of the continual adjustments with immediate visual feedback and temporal coherence that we provide.

Acknowledgments

We would like to thank our funding agencies Alberta Ingenuity, the Canada Foundation for Innovation, the Informatics Circle of Research Excellence, the Natural Sciences and Engineering Research Council of Canada, and SMART Technologies for their support, the artists for their comments, and Petra Neumann, Pauline Jepp, Uta Hinrichs, and Annie Tat for their advice.

References

BAXTER, B., SCHEIB, V., LIN, M. C., AND MANOCHA, D. 2001. DAB: Interactive Haptic Painting with 3D Virtual Brushes. In *Proc. of SIGGRAPH 2001*, ACM SIGGRAPH, 461–468.

BAXTER, W., WENDT, J., AND LIN, M. C. 2004. IMPaSTo: A Realistic, Interactive Model for Paint. In *Proc. of NPAR 2004*, ACM Press, New York, 45–56.

COLLOMOSSE, J. P. 2006. Supervised Genetic Search for Parameter Selection in Painterly Rendering. In *Applications of Evolutionary Computing*, Springer-Verlag, Berlin, 599–610.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K. W., AND SALESIN, D. H. 1997. Computer-Generated Watercolor. In *Proc. of SIGGRAPH 1997*, ACM Press, New York, 421–430.

DEUSSEN, O., HILLER, S., VAN OVERVELD, C., AND STROTHOTTE, T. 2000. Floating Points: A Method for Computing Stipple Drawings. *Computer Graphics Forum 19*, 3 (Aug.), 40–51.

DI BLASI, G., AND GALLO, G. 2005. Artificial Mosaics. *The Visual Computer 21*, 6 (July), 373–383.

ELBER, G., AND WOLBERG, G. 2003. Rendering Traditional Mosaics. *The Visual Computer 19*, 1 (Mar.), 67–78.

GOOCH, B., AND GOOCH, A. A. 2001. *Non-Photorealistic Rendering*. A K Peters, Ltd., Natick.

GOSSETT, N., AND CHEN, B. 2004. Paint Inspired Color Mixing and Compositing for Visualization. In *Proc. of InfoVis 2004*, IEEE Computer Society, Los Alamitos, 113–117.

GRUNDLAND, M., GIBBS, C., AND DODGSON, N. A. 2005. Stylized Rendering for Multiresolution Image Representation. In *Proc. of Human Vision and Electronic Imaging X*, SPIE/IS&T, Bellingham, 280–292.

HAEBERLI, P. 1990. Paint By Numbers: Abstract Image Representations. *ACM SIGGRAPH Computer Graphics 24*, 3 (Aug.), 207–214.

HAUSNER, A. 2001. Simulating Decorative Mosaics. In *Proc. of SIGGRAPH 2001*, ACM Press, New York, 573–580.

HAYS, J., AND ESSA, I. 2004. Image and Video Based Painterly Animation. In *Proc. of NPAR 2004*, ACM Press, New York, 113–120.

HERTZMANN, A., AND PERLIN, K. 2000. Painterly Rendering for Video and Interaction. In *Proc. of NPAR 2000*, ACM Press, New York, 7–12.

HERTZMANN, A. 1998. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. In *Proc. of SIGGRAPH 1998*, ACM Press, New York, 453–460.

HERTZMANN, A. 2003. A Survey of Stroke-Based Rendering. *IEEE Computer Graphics and Applications 23*, 4 (July/Aug.), 70–81.

HOPKINS, D. 1991. The Design and Implementation of Pie Menus. *Dr. Dobb's Journal of Software Tools 16*, 12 (Dec.), 16–26, 94.

ISENBERG, T., MIEDE, A., AND CARPENDALE, S. 2006. A Buffer Framework for Supporting Responsive Interaction in Information Visualization Interfaces. In *Proc. of C^5 2006*, IEEE Computer Society, Los Alamitos, 262–269.

KALNINS, R. D., MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., LEE, J. C., DAVIDSON, P. L., WEBB, M., HUGHES, J. F., AND FINKELSTEIN, A. 2002. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. *ACM Transactions on Graphics 21*, 3 (July), 755–762.

KEEFE, D. F., ACEVEDO FELIZ, D., MOSCOVICH, T., LAIDLAW, D. H., AND LAVIOLA JR., J. J. 2001. CavePainting: A Fully Immersive 3D Artistic Medium and Interactive Experience. In *Proc. of I3D 2001*, ACM Press, New York, 85–93.

KOWALSKI, M. A., MARKOSIAN, L., NORTHRUP, J. D., BOURDEV, L., BARZEL, R., HOLDEN, L. S., AND HUGHES, J. F. 1999. Art-Based Rendering of Fur, Grass, and Trees. In *Proc. of SIGGRAPH 1999*, ACM Press, New York, 433–438.

LITWINOWICZ, P. 1997. Processing Images and Video for an Impressionist Effect. In *Proc. of SIGGRAPH 1997*, ACM Press, New York, 407–414.

MARKOSIAN, L., MEIER, B. J., KOWALSKI, M. A., HOLDEN, L. S., NORTHRUP, J. D., AND HUGHES, J. F. 2000. Art-based Rendering with Continouous Levels of Detail. In *Proc. of NPAR 2000*, ACM Press, New York, 59–64.

MASON, K., DENZINGER, J., AND CARPENDALE, S. 2005. Negotiating Gestalt: Artistic Expression by Coalition Formation between Agents. In *Proc. of Smart Graphics 2005*, Springer-Verlag, Berlin, 103–114.

MEIER, B. J., SPALTER, A. M., AND KARELITZ, D. B. 2004. Interactive Color Palette Tools. *IEEE Computer Graphics and Applications 24*, 3 (May/June), 64–72.

MEIER, B. J. 1996. Painterly Rendering for Animation. In *Proc. of SIGGRAPH 1996*, ACM Press, New York, 477–484.

OLSEN, S. V., MAXWELL, B. A., AND GOOCH, B. 2005. Interactive Vector Fields for Painterly Rendering. In *Proc. of GI 2005*, A K Peters, Ltd., Wellesley, MA, USA, 241–247.

PARK, Y. S., AND YOON, K. H. 2004. Adaptive Brush Stroke Generation for Painterly Rendering. In *Proc. of Eurographics 2004, Short Presentations*, EUROGRAPHICS, Aire-la-Ville, Switzerland, 65–68.

RYOKAI, K., MARTI, S., AND ISHII, H. 2004. I/O Brush: Drawing with Everyday Objects as Ink. In *Proc. of CHI 2004*, ACM Press, New York, 303–310.

SALISBURY, M. P., ANDERSON, S. E., BARZEL, R., AND SALESIN, D. H. 1994. Interactive Pen-and-Ink Illustration. In *Proc. of SIGGRAPH 1994*, ACM Press, New York, 101–108.

SALISBURY, M. P., WONG, M. T., HUGHES, J. F., AND SALESIN, D. H. 1997. Orientable Textures for Image-Based Pen-and-Ink Illustration. In *Proc. of SIGGRAPH 1997*, ACM Press, New York, 401–406.

SCHKOLNE, S., PRUETT, M., AND SCHRÖDER, P. 2001. Surface Drawing: Creating Organic 3D Shapes with the Hand and Tangible Tools. In *Proc. of CHI 2001*, ACM Press, New York, 261–268.

SCHLECHTWEG, S., GERMER, T., AND STROTHOTTE, T. 2005. RenderBots—Multi Agent Systems for Direct Image Generation. *Computer Graphics Forum 24*, 2 (June), 137–148.

SECORD, A. 2002. Weighted Voronoi Stippling. In *Proc. of NPAR 2002*, ACM Press, New York, 37–44.

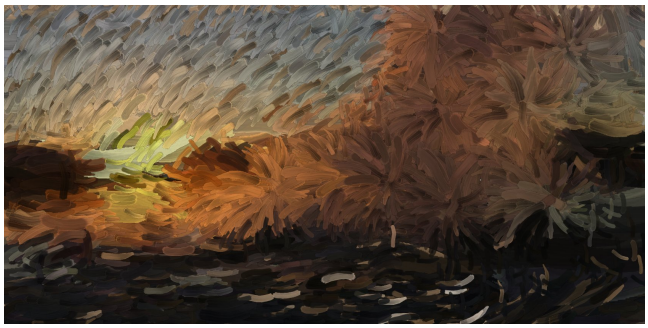
SEIMS, J. 1999. Putting the Artist in the Loop. *ACM SIGGRAPH Computer Graphics 33*, 1 (Feb.), 52–53.

SMITH, A. R. 1984. Plants, Fractals, and Formal Languages. *ACM SIGGRAPH Computer Graphics 18*, 3 (July), 1–10.

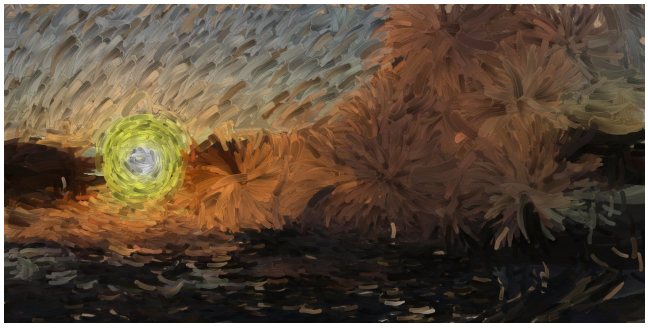
STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-Photorealistic Computer Graphics. Modeling, Animation, and Rendering*. Morgan Kaufmann Publishers, San Francisco.

YANG, H.-L., AND YANG, C.-K. 2006. A Non-Photorealistic Rendering of Seurat's Pointillism. In *Advances in Visual Computing, Part 2*, Springer-Verlag, Berlin, 760–769.

(b) Background layer.



(c) Middle layers added.



(d) Foreground layers added.

Figure 1: *Walking through the creation of an example image.*

Figure 9: *Wider strokes create an abstraction effect.*

Figure 10: *Longer strokes bring out the water reflections well; pointillism was used to portray the land part.*

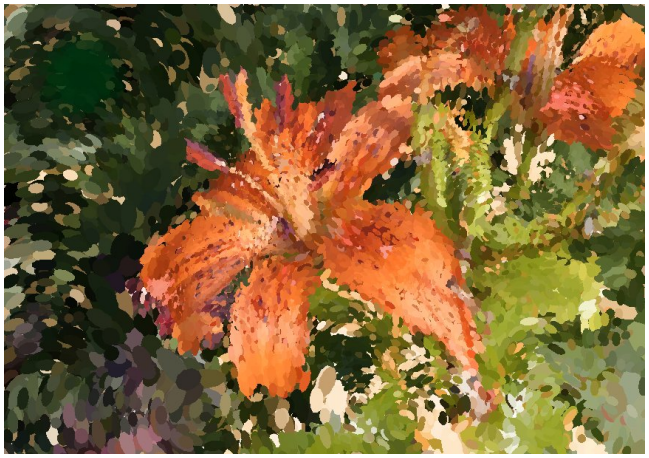


Figure 11: *Pointillism using simple geometric shapes.*

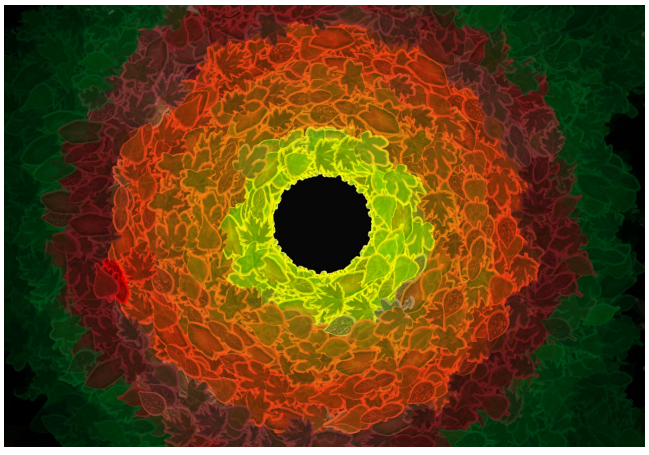


Figure 13: *Free-hand painting with leaves as primitives.*