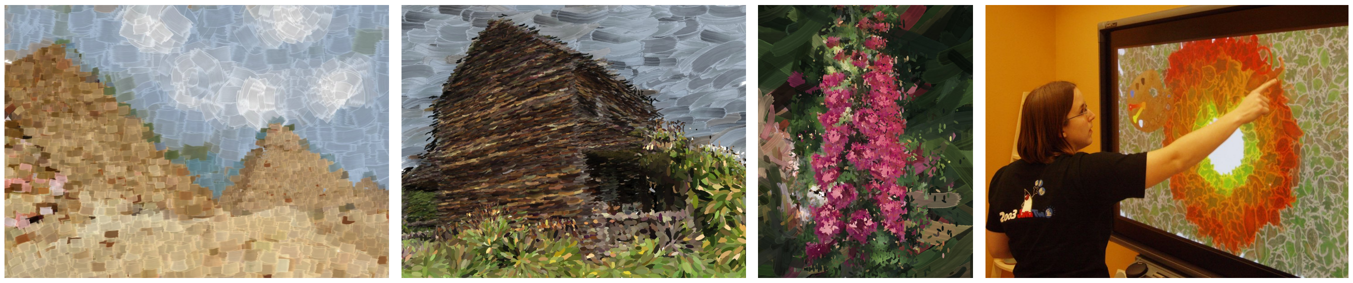
Modeling with Rendering Primitives: An Interactive Non-Photorealistic Canvas

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



Abstract

We present an interactive approach to non-photorealistic rendering that contrasts with the standard black box character of previous rendering techniques in that observation and interaction take place during rendering. Our technique is based on the idea of approaching non-photorealistic rendering by *modeling with rendering primitives*. This new approach supports interruption, tweaking, manipulation, and re-direction of the rendering as it develops. While we draw upon computational support for primitive placement to avoid having to painstakingly place each pixel, we limit the computational influence to enable freedom of interaction with the elements. We implement this new paradigm in a stroke-based rendering application using a stack of interaction buffers to store attributes of the primitives during the rendering. By manipulating the data in these buffers we affect the behavior of strokes on the canvas. This allows us to create and adjust images in non-photorealistic styles such as painterly rendering, pointillism, and decorative mosaics at interactive frame rates.

CR Categories: I.3.6 [Computer Graphics]: Methodology and Techniques—Interaction Techniques; I.3.8 [Computer Graphics]: Applications; I.3.m [Computer Graphics]: Miscellaneous—Non-Photorealistic Rendering

Keywords: Non-photorealistic rendering (NPR), interaction techniques, modeling with rendering primitives.

1 Introduction

As the field of computer graphics matures, the capabilities to create rich and subtle digital visuals continue to expand. These range from increasingly realistic results 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 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.

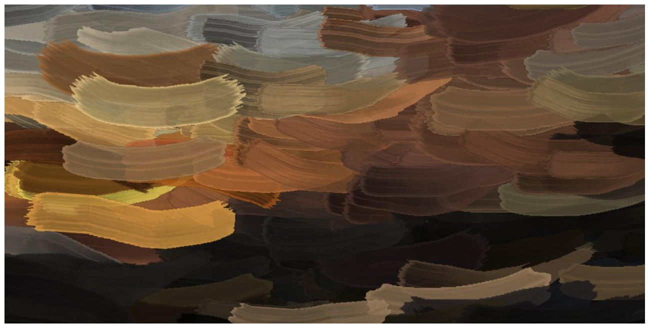
While one could argue that any amount of richness is available if one is prepared to place and color each pixel, realistically this interaction needs to happen on a more meaningful level. One example for this is the use of common artistic primitives such as brush strokes because they represent elements 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 correct simulations of traditional techniques. We contribute a new paradigm for interactive image creation in which an image is constructed with pre-rendered primitives through an interactive process that can call upon computational support rather than being controlled by it. We think of this process of image construction as *modeling with rendering primitives*. We illustrate the new paradigm 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 being created instead of being "cast in stone."

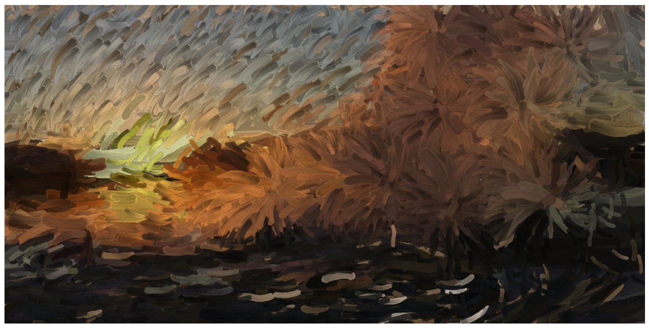
The new possibilities for interacting with the rendering process enable a wide variety of ways to create images. Here is an example illustrating a possible usage of our system (Figure 1). First, we start by creating the background. In order not to have to paint the

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

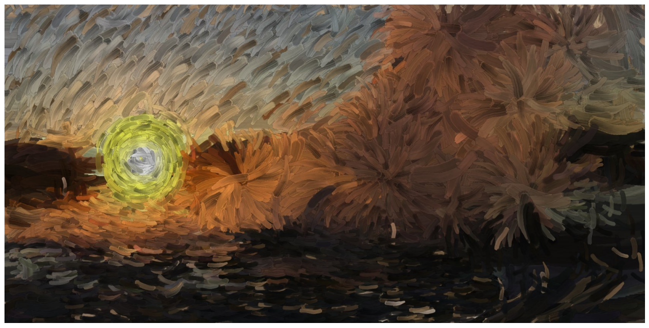
(a) Source image to derive color distribution.



(b) Background layer.



(c) Middle layers added.



(d) Foreground layers added.

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

color for each layer we load a source image to be used as a base color distribution (Figure 1(a)). Next, we fill this first (background) layer with strokes. The strokes created are relatively large and give the general color impression as well as direction of painting (Figure 1(b)). For the middle layers, the source image is used again to determine the color distribution. Here, smaller strokes are used to give distinct regions more detail (Figure 1(c)). For example, the strokes in the sky were given one unified direction and the steam was portrayed using radial orientation in several places to work out its dynamic character. This example nicely 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 a clear sky [Park and Yoon 2004]. Finally, we create foreground layers capturing the highest level of detail. Small, fine strokes are added to those layers (Figure 1(d)). For example, we use a circular orientation effect to emphasize the sun's round shape. The color distribution was modified at all three stages by painting over it where it seemed desirable to touch up regions or add color effects. In this example we have used the same source image as a reference for all three layers, although this is not necessary. It is possible to use different degrees of Gaussian blur on the same image for the different layers or use different images altogether to achieve a specific effect.

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

To contextualize this research with previous literature, we describe recent digital painting systems, mention related NPR techniques, and then discuss existing NPR interaction paradigms.

Digital painting systems (see [Smith 2001] for a historical overview) are very heavily employed today in the creation of digital art. There are two major categories of systems: pixel-based (e. g., Adobe® PhotoShop®, Corel® Paint Shop Pro®, TwistedBrush, and Gimp) and vector-based (e. g., Adobe® Illustrator®, CorelDraw®, and Inkscape). Systems in the first category 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. The latter representation provides more freedom for subsequent changes than the first because primitives are stored using their properties as opposed to being in rasterized form. With respect to the representation of primitives, our technique and system can be thought of as a hybrid of these two approaches. From the pixel-based approach, we use painting interaction techniques and rasterized primitives, but we maintain all our primitives as actively adjustable, as with vector approaches. In addition, we use painting interaction to manipulate properties of primitives instead of applying filters locally.

Our work makes use of higher-level primitives as developed in non-photorealistic rendering 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], graftals [Smith 1984; Kowalski et al. 1999; Markosian et al. 2000], 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]. Most of these approaches strive to simulate traditional techniques of artistic expression or illustrative depiction. Some of these styles have been used by artists to work around limitations of the chosen medium (e. g., stippling and hatching to represent gray scales in printing). The computer has provided new ways to push the boundaries of possible depiction methods, for example, developing dynamic primitives such as graftals [Kowalski et al. 1999; Markosian et al. 2000],

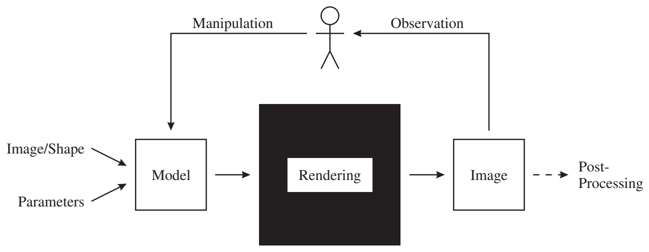


Figure 2: Traditional rendering: the rendering phase acts like a black box, prohibiting direct interaction with the process.

which can algorithmically generate geometry in the rendering process depending on the current view and other parameters.

While some NPR techniques are probing the possibilities of increased interaction, most are still limited in the options provided. An example of increased interactivity is the WYSIWYG-NPR system [Kalnins et al. 2002] that allows changing the rendering style by painting example strokes, which are then used as a template for the algorithmic creation of all the strokes in the image. In a related approach, Salisbury et al. [1997] allow users to paint the orientation of hatching strokes onto a 3D model. Deussen et al. [2000] used brushes to interactively manipulate the relaxation of stipple points as well as create new ones or delete them.

Most closely related to our work in terms of interactivity are Haeberli's Paint by Numbers [1990], the RenderBots system [Schlechtweg et al. 2005], and Negotiating Gestalt [Mason et al. 2005]. Paint by Numbers used a source image and a canvas. One could 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 freedom, it was not possible to change the strokes or their properties after they had been placed. RenderBots combine a multi-agent system with NPR rendering. Here, a user can brush autonomous agents (RenderBots) onto the canvas, which then read values from a stack of G-buffers (created in pre-processing) 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 RenderBots' actions. Negotiating Gestalt uses a multi-agent system to model the image creation process as coalition forming. It allows one to take direct control of agent or coalitions during the process. This is a step away from the typical 'black box' rendering approach; however, interaction via agents is still indirect.

3 Modeling with Rendering Primitives

Our goal is to support interruption, tweaking, manipulation, and redirection of the rendering as it develops. We make this possible by constructing a model with rendering primitives so that once the construction is achieved the rendering is also complete. A parallel can be drawn between this process and that of an artist creating a painting out of individual brush strokes. However, while an algorithm should not make parts of the process inaccessible, neither should it be necessary to painstakingly and manually place each rendering primitive. We intend to draw upon computational support for placement and adjustment of primitives while maintaining the possibility of interaction during rendering.

Most techniques in NPR are roughly based on the following process for generating images (see Figure 2). First, a model is constructed. For our discussion, this model could be many things such as constructive solid geometry, a three-dimensional polygonal mesh with its surface properties, a volume data structure, or a 2D digital image. Choosing, developing, and generally working with the model are usually possible. Once the model is ready to be rendered, there are many choices of rendering methods, each method having many possible parameters. After these choices are made, the rendering proceeds. While in many cases the rendering process is not overly long, it will proceed until it is completed. After the resulting image has been created, once again there are a myriad of interaction possibilities in terms of post-processing. If the results is not fully satisfying, it is possible to revert to the modeling stage and restart the rendering with new parameters. While some of the NPR techniques, such as those that operate on 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, which can then be subjected to post-processing.

In general, interactive input is only possible in the first and the last stage of this pipeline because the algorithmic processing has been defined by the programmer. Even though the algorithm may have been artistically inspired, it can typically only be controlled by setting its parameters. By turning the rendering into an interactive process some effects become more accessible. Examples include integrating more than one rendering style, tweaking brush strokes that are not aligned with the intended artistic impact, finding brush stroke arrangements where algorithmic approaches have trouble such as unstructured sky regions, or where it is necessary to experiment with how rendering primitives are placed. Thus, creative freedom at all stages of the image production process is a worthwhile goal.

To make it possible to model with rendering primitives it is necessary to both have an understanding of what is meant by rendering primitives, and have a method by which it is possible to provide non-controlling computational support for the process of modeling with them. As stated before, our goal is to provide interactive control over the process as a whole to provide freedom in image generation. This cannot be achieved simply by making the algorithmic aspect of the rendering stage more accessible by, for example, increasing the number of parameters. Nor does this mean that algorithmic support should be removed altogether. Instead, computational support for influencing the rendering process should be flexibly available, leaving one free to choose the degree to which one will use it.

In our terms, a rendering primitive is simply a part of an image. In its simplest computational form, a rendering primitive can be a pixel—a single cell in a rasterized digital image. To be usable in practice, a rendering primitive can be any small part from which images can be created, including all sorts of brush strokes, such as lines, points, dabs, dots, mosaic tiles, other images, parts of images, or any visual aspect from which another image can be created. While the ability to create rendering primitives interactively is important to our approach, we make use of existing methods in current paint systems as well as NPR approaches for physical simulation of traditional media.

Therefore, starting from a collection of rendering primitives, we approach the next problem: how to provide algorithmic support for the modeling stage as an image is created from rendering primitives. Once again there are interesting ideas in the literature from which can be extrapolated. In particular, we consider Paint by Numbers [Haeberli 1990] in which it is possible to create a differently-styled version of a source image by iteratively painting on it. A second source image may be used to influence the stroke style. While this is, in effect, interactively creating a rendering, the possibilities are limited in that only new stylizations of the source images are possible. Therefore, we consider the input images to be spatial informa-

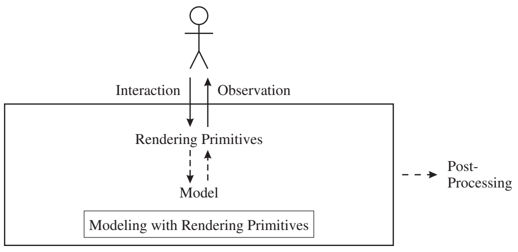
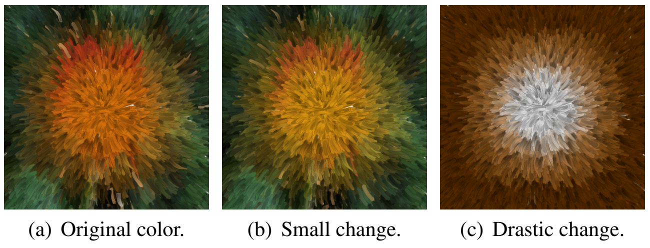


Figure 3: Modeling with rendering primitives.



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

Figure 4: Affecting the color of primitives.

tion that can be stored as buffers, and make these buffers interactive. This leads to truly being able to take control of the rendering process. We expand on this idea, integrating it with ideas of click-less interaction and complex adaptive systems to create an interactive environment where it is possible to model with rendering primitives.

Figure 3 shows a diagram of the changed pipeline for our system. By placing the rendering primitives and model into the interactive stage, it is now possible to both observe what is happening and work directly within that stage. Starting with a general concept, the model can be built and adjusted as the work progresses. Since the primitives are fully rendered, the image takes form though the interaction. This concurrent modeling and rendering is essential: the model is created and constantly updated with rendering primitives as they are generated, manipulated, and removed.

4 An Interactive NPR Canvas

To enable modeling with rendering primitives, our system will require: (1) a mechanism that can efficiently support simultaneous manipulation of many primitives, (2) tools to assign meaningful values and useful interactions, (3) an interface to coordinate the tools, and (4) efficient rendering techniques to maintain interactive frame rates. Since the challenge lies in providing interactive support of a multitude of primitives, we focus our discussion there.

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 as 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 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. Depending on this attenuation, the color tool mixes the new color with the previous color values read from the color buffer (Figure 4). 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. Similar to the color tool, the size tool changes the size of objects by reducing or increasing the local buffer values (Figure 5). The

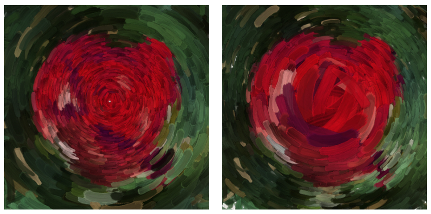

(a) Strokes smaller. (b) Strokes larger.

Figure 5: Resizing elements.

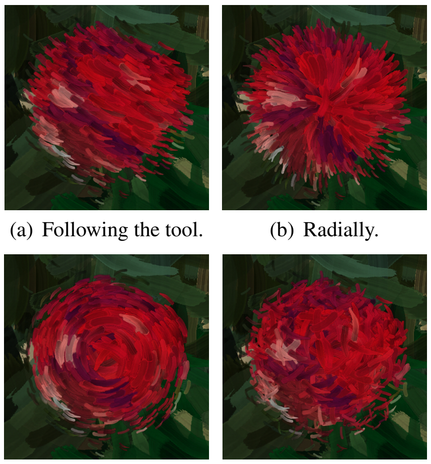

(a) Following the tool. (b) Radially.
(c) Circumpolar. (d) Randomly.

Figure 6: Changing primitive orientations interactively.

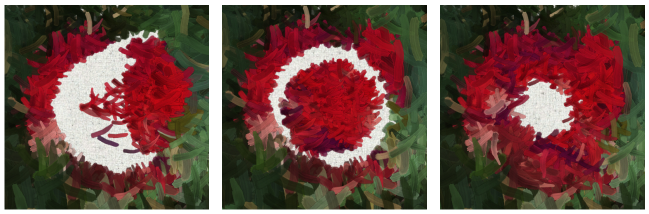

(a) Following the tool. (b) Attraction. (c) Repulsion.

Figure 7: Circular motion tool affects the position of primitives.

orientation and motion tools both have effects either independent from or dependent on the motion of the tool while it is being used (Figures 6 and 7). The orientation tool changes the orientation to follow its motion path (Figure 6(a)) or imposes motion-independent orientations (Figures 6(b) to 6(d)). Likewise, the motion tool forces primitives to follow the movement of the tool (Figure 7(a)) or exerts an attraction (Figure 7(b)) or repulsion (Figure 7(c)) force on primitives with respect to the tool's current position. The eraser tool slowly erases objects from the canvas (Figure 8). It renders probabilities for primitives to delete themselves into the erase buffer. 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. 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. One final tool that has a special behavior is the tool to create new strokes. It creates new primitives at its location instead of rendering into a property buffer.

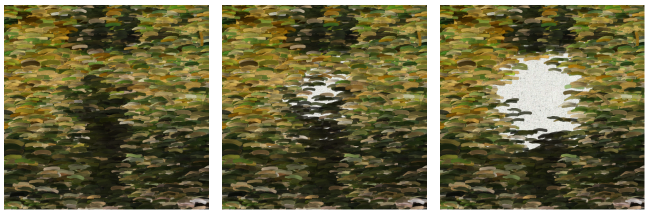

(a) Original. (b) Few strokes erased. (c) Continued erasing.

Figure 8: Gradually erasing elements by means of probabilities.

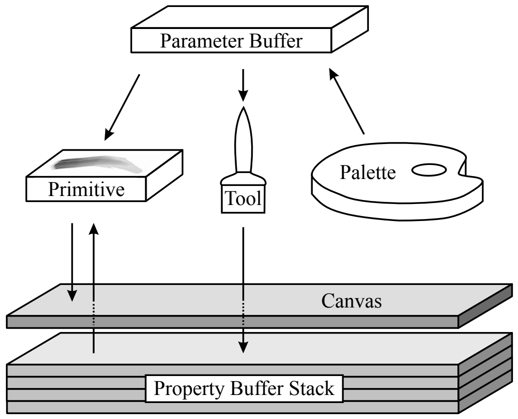


Figure 9: Communication between interface elements.

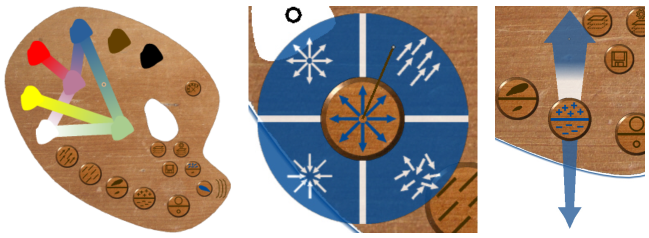

(a) Color mixing. (b) Pie menu. (c) Slider.

Figure 10: Palette as a central control.

4.3 Interface Elements

In addition to the tools described above, our interface consists of two more elements: (1) a canvas, where primitives are placed and which holds the property buffers and (2) a palette to control the tools, to specify what primitive properties to manipulate and with what values. The communication between these is realized using a parameter buffer (Figure 9) that maintains system state and 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 10) provides means for tool selection and parameter specification. It writes the appropriate values into the parameter buffer where tools can access them. The palette affords color mixing (Figure 10(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 the original color to the mixed one is shown. Colors can now be chosen from any point on the color blobs or the gradients between them. The other interface elements on the palette are represented by buttons, pie menus (Figure 10(b); [Hopkins 1991]), and sliders (Figure 10(c)) to enable click-less operation using only touch and move interactions.

4.4 Efficient Rendering

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 million pixels) as well as on a large (146 cm × 110 cm), high-resolution (2,800 × 2,100 pixels; 5.88 million 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. In the desktop setting, both cards were combined in SLI mode (without bridge) to drive one screen; in the table setting, the machine drove 4 projectors (2 × 2, with simulated fullscreen mode). Table 1 gives timings for both

stroke	stroke textures		pointillism	
count	desktop	table	desktop	table
1,000	177	29	159	31
2,000	96	24	86	27
4,000	51	19	32	21
8,000	26	13	16	15
16,000	13	8	8	10

Table 1: Timings in fps for both settings using 256^2 mip-mapped textures and pointillism blobs, each covering about 128^2 pixels.

settings that include all strokes updating from the buffer stack. We noticed that neither rendered primitives size nor the texture size affected the frame rates much on this setup, while we saw differences on other hardware. We were able to fill the entire tabletop screen with about 4,000 strokes, each covering about 128^2 pixels; for the desktop setting, 1,000 strokes sufficed. For creating the images in this paper, we used up to 16,000 strokes in several layers, which we were able to display at interactive rates since not all were reading from the buffer stack at the same time.

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.

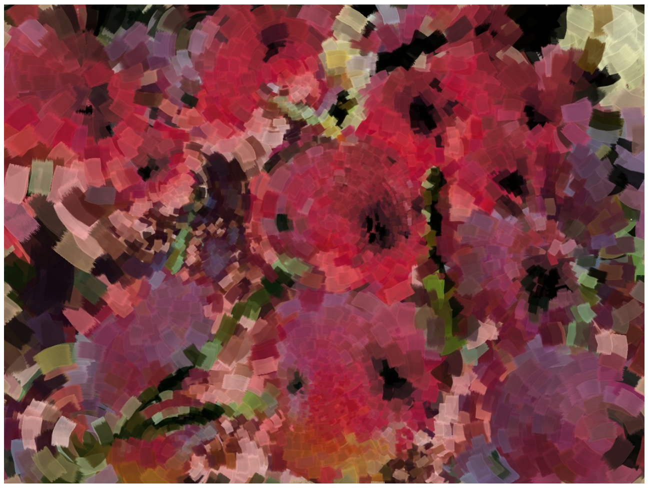


Figure 11: Wider strokes create an abstraction effect.

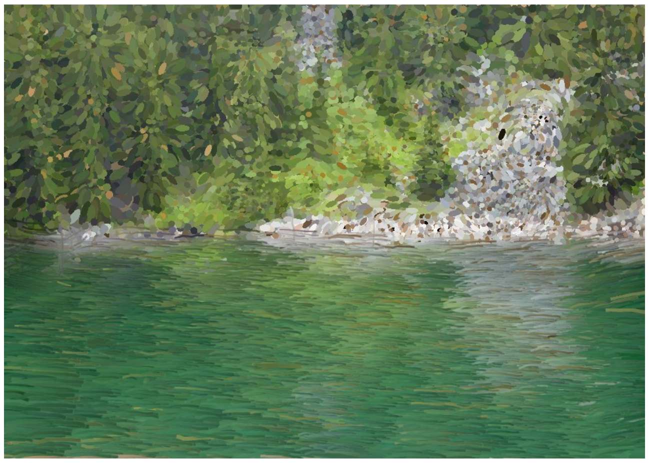


Figure 12: Longer strokes bring out the water reflections well as well as pointillism to portray the land part.

5.1 Example Images

Figure 11 shows how to use quite broad stroke textures 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 12 works with very long and thin strokes as they capture the structure of the small waves on the water and, thus, the character of the reflection very well. The strokes were arranged parallel to the water line to produce this effect. Since such long strokes caused undesirable artifacts in the background, we used pointillism strokes for this region instead. Figure 13 only makes use of the 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 14 uses four differently colored versions of the same source image as background and teapot shapes as primitives. Figure 15 shows the use of mosaic tiles that have been manually oriented to indicate groups of the surrounding water plants as well

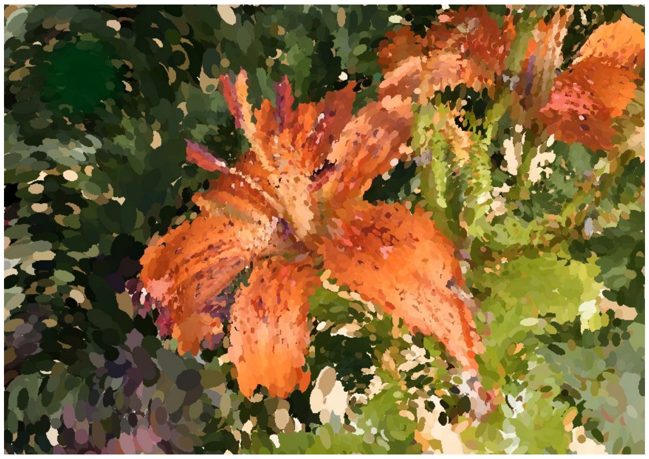


Figure 13: Pointillism using OpenGL shapes.

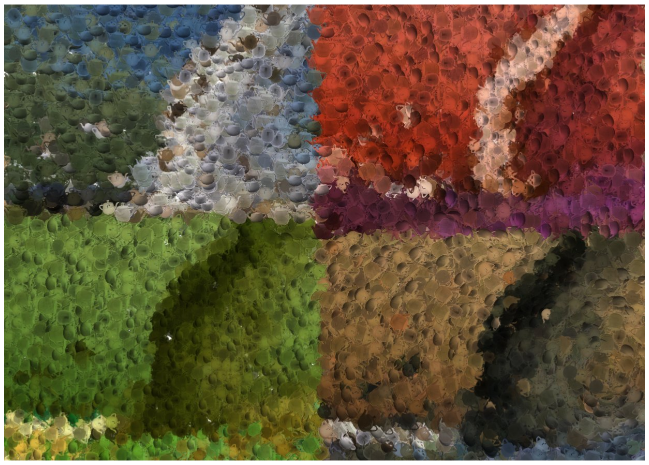


Figure 14: Teapot collage.

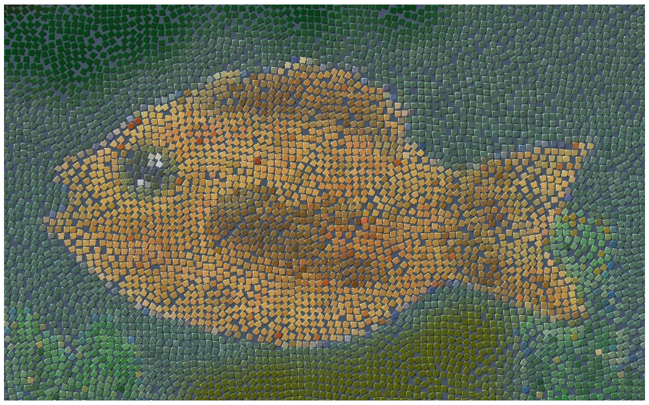


Figure 15: Decorative mosaics, tiles with shading effects.

as features of the fish itself. Finally, Figure 16 shows a painting created without a source image using colored leaves as primitives. The motif was inspired by works by Andy Goldsworthy Rowan.

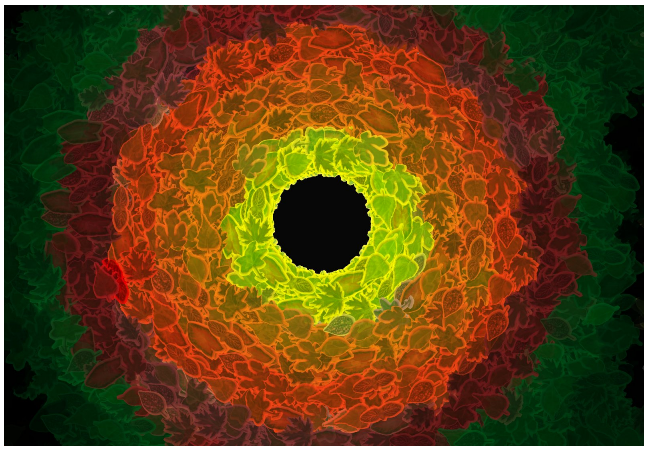


Figure 16: Free-hand painting with leaves as primitives.

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 noise present in the color distribution of the loaded image as well as the textures of the strokes. Artists were intrigued by the effect that if single strokes were tossed across the canvas they 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 is an aspect that 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 new paradigm of painting. One comment was, for example, that the new paradigm feels *"something like trying to compose a picture with feathers that keep moving about in the air."* However, after getting used to the new paradigm, people liked that it was very easy to create images in a very short time.

We also received feedback from the artists about the interface. This led us to change the interface several times, incorporating their input to address their concerns. For example, the use of layers to separate regions from one another as well as the ability to temporally 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 developed a new paradigm for creating and interacting with non-photorealistic rendering. By opening up the black box of rendering we make it possible to influence the primitives as they are assembled on the canvas and to modify aspects as desired. *Modeling with rendering primitives* opens up entirely new 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 new paradigm and explored how it enables artists to take control of the image production process.

Our concept opens up new possibilities for building tools that enable interaction directly with the rendering process. Thus, our system can be extended in a number of ways. As the interaction with primitives through buffers is indirect, a more direct way would be to only have instantaneous buffers to change properties of elements. Even though this would remove an interesting aspect of the system (that primitives change as they are moved across the buffers) this may lead to a more direct form of interaction with the primitives. Certainly, one could explore a more diverse set of tools, representing other primitive properties, as well as more powerful primitives such as graftals to increase the power of expression. In this context, it would also be challenging to incorporate larger primitives, rigid or flexible, that have more than one point to query property buffers. Similarly it would be interesting to load images into buffers other than the color buffer, without limitations to G-buffers [Schlechtweg et al. 2005] and to allow tools to do image processing on the buffers.

As we developed the system to also be applicable to large displays with multi-touch input, it would be very interesting to explore the use of multiple inputs in the interaction such as multi-finger painting. We are also interested in investigating the difference of use of our system between traditional desktop environments and 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.

References

DEUSSEN, O., HILLER, S., VAN OVERVELD, C. W. A. M., 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, CA, 113–117.

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.

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.

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.

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.

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.

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.

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.

SMITH, A. R. 2001. Digital Paint Systems: An Anecdotal and Historical Overview. *IEEE Annals of the History of Computing 23*, 2 (Apr.–June), 4–30.

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, Heidelberg, 760–769.