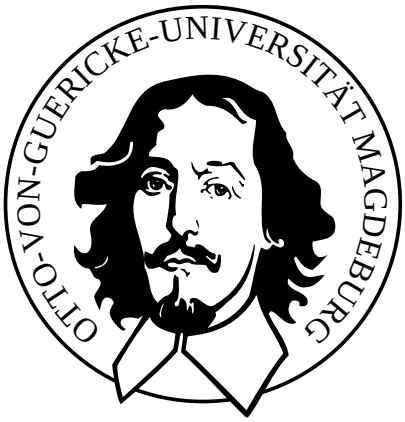
# AN INTERACTIVE NON-PHOTOREALISTIC CANVAS

Studienarbeit by Martin Schwarz




Department of Simulation and Graphics
Faculty of Computer Science
Otto-von-Guericke-University of Magdeburg

April 2007

SUPERVISORS:
Prof. Dr. Sheelagh Carpendale
Dr.-Ing. Tobias Isenberg
Dipl.-Ing. Tobias Germer

## ABSTRACT

Within the area of non-photorealistic rendering, many techniques have been developed to simulate a wide variety of different styles. However, these styles are usually globally applied to some form of model that is provided as input and specific local interaction is often not possible. This thesis explores the application of the paradigm of modeling with rendering primitives to enable users to fully influence the creation of digital imagery in various artistic styles.

The application of this paradigm offers a more flexible and more interactive way of working with image components. The implemented system provides tools for the manipulation of a large number of primitives at once. The thesis and the implementation described aim at enabling new possibilities of interacting with digital paintings that allow more local control and artistic freedom within the creation process of digital imagery. As such the approach is a hybrid technique. It combines the advantages of visually rich but inflexible pixel graphics on the one hand and visually less rich but more flexible vector graphics on the other hand, and adds new possibilities to allow fluid interaction.

To realize this concept, image components have been used that are easy to manipulate by offering many of the flexibilities of vector primitives while still maintaining the visual quality of pixel graphics. The interaction is facilitated by employing a buffer approach that retains a responsive interaction even with a large number of these primitives. In addition, a user interface has been developed that combines all the functionality on one compact widget. It is also applicable to large, interactive displays where features such as click-less interfaces are necessary.

The presented system allows the users to create non-photorealistic images without being constrained by an encapsulated algorithm. Also, it provides a new way to explore painting by permitting a high degree of local control while still maintaining some degree of algorithmic support. This enables users to easily create expressive images and interesting effects.

# ZUSAMMENFASSUNG

In dem Gebiet des nicht-fotorealistischen Renderings wurden viele Techniken entwickelt, um eine Vielzahl unterschiedlicher Stile zu simulieren. Jedoch werden diese Stile für gewöhnlich global auf ein bestimmtes Modell, das als Eingabe dient, angewendet und eine gezielte lokale Interaktion ist meistens nicht möglich. Diese Studienarbeit untersucht die Anwendung des Paradigmas "modeling with rendering primitives", um den Anwendern zu ermöglichen, die Erzeugung von digitalen Bildern in verschiedenen künstlerischen Stilen gänzlich zu beeinflussen.

Die Anwendung dieses Paradigmas bietet eine flexiblere und interaktivere Art des Arbeitens mit den Komponenten in einem Bild. Das implementierte System stellt Werkzeuge für die gleichzeitige Manipulation einer großen Anzahl von Primitiven bereit.

Die Studienarbeit und die darin beschriebene Implementierung zielen auf die Erzeugung neuer Möglichkeiten für die Interaktion mit digitalen Bildern ab, die mehr lokale Kontrolle und künstlerische Freiheit während der Erzeugung von digitalen Bildern erlauben. Als solches ist dieser Ansatz eine hybride Technik. Sie verbindet die Vorteile von visuell ansprechenden aber unflexiblen Pixel-Grafiken auf der einen Seite und die visuell weniger ansprechenden aber flexibleren Vektor-Grafiken auf der anderen Seite, und lässt neue Möglichkeiten für eine flüssige Interaktion zu.

Um dieses Konzept umzusetzen, wurden Bildkomponenten verwendet, welche einfach zu manipulieren sind, indem sie viele Freiheiten von Vektor Primitiven bieten, und dabei immer noch die visuelle Qualität von Pixel-Grafiken behalten. Die Interaktion wurde ermöglicht durch den Einsatz eines Pufferansatzes, der eine leistungsfähige Interaktion auch mit einer großen Anzahl dieser Primitive gewährleistet. Zusätzlich wurde eine Benutzerschnittstelle entwickelt, die alle Funktionalitäten in einem kompakten Widget vereint. Sie ist auch auf großen, interaktiven Bildschirmen einsetzbar, auf denen andere Feature, wie zum Beispiel die Eingabe ohne Klick notwendig sind.

Das vorgestellte System erlaubt den Anwendern nicht-fotorealistische Bilder zu erzeugen, ohne von einem verborgenen Algorithmus eingeschränkt zu sein. Es gestattet auch eine neue Art des Malens, indem ein hoher Grad an lokaler Kontrolle ermöglicht und trotzdem ein gewisser Anteil an algorithmischer Unterstützung geboten wird. Dies erlaubt Benutzern mit einfachen Mitteln ausdrucksvolle Bilder und interessante Effekte zu erzeugen.

## PUBLICATIONS

Some ideas and figures have appeared previously in the following publications:

Schwarz, M., Isenberg, T., Mason, K., and Carpendale, S. 2007. Modeling with Rendering Primitives: An Interactive Non-Photorealistic Canvas. Tech. Rep. 2007-851-03, Department of Computer Science, University of Calgary, Canada, Feb.

Schwarz, M., Isenberg, T., Mason, K., and Carpendale, S. 2007. Modeling with Rendering Primitives: An Interactive Non-Photorealistic Canvas. In Maneesh Agrawala and Oliver Deussen, editors, *Proceedings of the Fifth International Symposium on Non-Photorealistic Animation and Rendering* (NPAR 2007, August 4–5, 2007, San Diego, California, USA), New York, 2007. ACM Press. Submitted.

Tobias Isenberg, Simon Nix, Martin Schwarz, André Miede, and Sheelagh Carpendale. Mobile Spatial Tools for Fluid Interaction. *In Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology* (UIST 2007, October 7–10, 2007, Newport, Rhode Island, USA), New York, 2007. ACM Press. Submitted.

# ACKNOWLEDGMENTS

I received a lot of help while working on this internship project and the subsequent thesis. In the Interactions Laboratory I was provided with an excellent work environment and surrounded by talented and smart researchers. I would especially like to thank the following people:

Tobias Isenberg for his supervision, lots of programming advice, and writing support.

Sheelagh Carpendale for inviting me to Calgary and the Interaction Laboratory and giving me this project. I would also like to thank her for her advice on the project and for her help on the paper.

André Miede for his work on the buffer framework and his beautiful classic thesis style.

Petra Neumann for painting a couple of very nice images and also for providing me with numerous photos from her Canada collection. I would also like to thank her for her help on the video that we made for the paper.

Annie Tat for reviewing the color mixing and for her tips on some particular icon designs.

Uta Hinrichs for tips on interface elements.

Andrew Seniuk for his mathematical help.

Torre Zuk for programming advice and speed-up tips.

Kimberly Tee for patiently posing at the tabletop and wall displays.

Edward Tse for providing me with photos from his beautiful collection, that i used as input images for stroke coloring.

Arlene Stamp for numerous comments and a lot of helpful advice on the program.

# CONTENTS

# LIST OF FIGURES

## LIST OF TABLES



## ACRONYMS

| | |
|---|---|
| NPR | non-photorealistic rendering |
| SBR | stroke-based rendering |
| WYSIWYG | What You See Is What You Get |
| WIMP | Windows, Icons, Menus, and Pointers |
| RNT | Rotate'N Translate |
| RYB | Red Yellow Blue |
| RGB | Red Green Blue |
| CMYK | Cyan Magenta Yellow Black |
| OpenGL | Open Graphics Library |
| dpi | dots per inch |
| fps | frames per second |
| DViT | Digital Vision Touch |

# 1 INTRODUCTION

There are a lot of non-photorealistic rendering (NPR) techniques nowadays that cover a vast number of artistic styles. For example, computers can simulate pencil drawings, watercolor paintings and cartoon-like effects. However, it is rarely possible to control the appearance of an illustration by making local changes. This is especially deleterious for an artistic utilization of such algorithms because they restrain the artistic freedom. Digital image applications should give as much control as possible to the users and allow them to easily interact with the process of creating a painting. This thesis tries to address this issue and describes a system that gives the control back to the users.

This first chapter presents the decisive factors for developing the system described in this thesis and gives a first impression of what has been accomplished with this work.

## 1.1 MOTIVATION

NPR is the endeavor of depicting and visualizing complex scenes in a way that communicates as effectively as possible [Strothotte and Schlechtweg 2002]. NPR brings together art and science and concentrates more on the communication or the content of an image. It can be thought of as subjective, whereas Photorealism is rather objective. It is inspired by imagery generated by artists and illustrators. This imagery usually provides information about objects that may not be readily apparent in photographs or real life. NPR techniques are able to produce still images or entire animations.

NPR research has developed a vast number of algorithms for the placement and alignment of image primitives. Brush strokes, for example, are scaled, bend, oriented and colored according to the input image or 3D model. But it remains difficult to find an appropriate orientation for the brush strokes in areas such as a uniform blue sky which has no evident orientation for strokes [Park and Yoon 2004]. Most work in NPR has focused on algorithms that are controlled by parameter settings or scripting. The users have no direct control over where marks are made. Little research has been done to get the users more involved and to allow image creation based on the users input. Computers can repeat tasks and produce fine detail, but without the users, simulating artistic expression is impossible.

Two kinds of representations have been established for non-photorealistic renderings. Pixel graphics have become a popular storage format for digital paintings and renderings in general. Although pixel graphics offer an amazing visual richness and are able to capture a vast amount of rendering effects, their manipulation can be very laborious or has to be carried out using global tools, such as filters, or more local tools, such as digital brushes. Vector graphics, on the other hand, offer more flexibility and primitives are not embedded and fixed in the

image plane. However, the manipulation of a large number of vector primitives is still a problem because it requires nonintuitive interaction techniques like marking or grouping objects together.

Another motivation for this work was to transfer digital painting from standard desktop PCs to large, high-resolution interactive displays. The two main advantages of such displays are the increased physical size while still offering a high resolution. Such a large display offers viewing of data at true-to-life or human-scale physical sizes [Ni et al. 2006]. Painting on a very large display with a similar size as a real canvas or even bigger creates an immersive and more familiar painting environment.

## 1.2 CONTRIBUTIONS

User input and rendering often occur in separate stages of the simulation process, often creating a mismatch between what the system delivers and what the user actually had in mind. This work aims to fill that gap. In order to immediately visualize the impact of manipulations to the image, the concept to have modeling and rendering occur in parallel has been applied to non-photorealistic rendering. The main focus is the possibility to interact with the painting, at all times, and full control over the image creation process. Instead of just having an algorithm controlled by a number of parameters, that creates the expected image, the research presented in this thesis focuses on the user's manipulation of the image.

This system presents a new and easy way to interact with image components. The tools embedded in the system and the practical application of a sophisticated buffer approach allow for easy modification and attribute change of image elements without the need to select or group them. This leads to an effective manner for manipulating a large number of primitives at once. Image elements presented in this thesis provide the visual quality of pixel graphics and offer a number of the flexibilities of vector graphics.

The above mentioned concepts have been applied to three common NPR techniques: *painterly rendering*, *decorative mosaics*, and *pointillism*. Imagery created with these artistic styles are usually composed out of a large number of primitives which makes them eligible for the approach used in the system described in this thesis and the use of tools to manipulate a lot of them at the same time. Images created with the system can be expressive and enable interesting new effects (see Figure 1 for example).

In addition, the implemented system can be used on regular desktop PCs, as well as, large, high-resolution displays. For the latter one, appropriate interaction metaphors have been implemented that allow direct manipulation of primitives on these interactive displays. The user interface has been adapted to the users' needs with the aid of participatory design and informal user tests.

Figure 1. Example image created with the system described in this thesis. Original photo © Petra Neumann used with permission.

## 1.3 LIMITATIONS

The system described in this thesis is not just another painting program, but it is instead an attempt to create a new way of interacting with the components of a painting. Also, it was not intended to recreate a complete palette of tools that is usually available in modern paint applications. One of the main goals of this work is to make working with a painting fully interactive at fast rendering rates. Therefore, an accurate simulation of natural media is not feasible and not the goal of this work. Special emphasis has been put into the user control over the overall process. That is why there are no fully automated rendering algorithms in this implementation.

## 1.4 ORGANIZATION

The thesis is structured as follows:

CHAPTER 2 gives an overview of the related work and thereby frames the thesis. It introduces terms and concepts used throughout the subsequent chapters. This chapter is essential to understand the implementation details given in Chapter 4. It especially examines interaction techniques and interface considerations needed for high-resolution displays.

CHAPTER 3 explains the underlying concepts of this thesis. It compares the traditional rendering process common in NPR with the con-

cept of concurrent modeling and rendering used in this thesis. Then, it introduces the buffer approach utilized to guarantee interactive painting even with a high number of primitives on the canvas. In the last part, the interaction between the major system components, as a vital basis for the system, is pointed out.

CHAPTER 4 deals with implementation details and the program structure. It provides low level details such as the description of the main classes of the program and optimizations that have been done to speed up computation time. It also provides the reader with the characteristics of the individual tools, their usage and effects.

CHAPTER 5 discusses the results and outcomes of this work. It presents a number of example images and gives a short evaluation of the deployment of the system on different display devices. It summarizes this thesis and completes it with an outlook into possible future work and extensions.

# 2 FUNDAMENTALS

In order to position this thesis among research in domains, such as, painting systems, non-photorealistic rendering and stroke-based rendering (SBR), an overview of related work is presented in this chapter. It also contains all the things needed to understand the terms and definitions used in the following chapters.

## 2.1 NON-PHOTOREALISTIC RENDERING

NPR researchers have presented a huge amount of solutions to reproduce artistic styles and render meaningful imagery. Some algorithms and findings can already be found in everyday graphics editors and digital painting applications.

### 2.1.1 *Digital Painting Systems*

One of the first interactive painting programs is PAINT [Smith 1978]. PAINT has a canvas, a brush, that is represented by a two-dimensional shape, and the user can pick a color or mix a new one. It has a considerably large functionality and is a pioneering endeavor in the field of painting applications. SMITH's further development, TABLE PAINT [1979], is an attempt to make painting more natural. He introduces the usage of an additional framebuffer, a stylus, which is to be used with a tablet, and an extra device as a controller to add a third dimension to the stylus. The user has the ability to change size, shape, orientation, and color. SMITH also says that painting must occur in real-time to be interesting.

In HAEBERLI's PAINT BY NUMBERS [1990] the user can paint brush strokes with the mouse and their color is determined by an underlying source image. Each click creates a brush stroke with a user specified location, shape or style, size and orientation. The mouse speed or the keyboard can be used to control the strokes size. The user receives a lot of support from the program while working with it. PAINT BY NUMBERS is a seminal system that also creates artistic styles such as decorative mosaics and pointillism, that are now common to NPR (see Figure 2).

A very promising related work is the COOLPAINT system by LANG et al. [2003]. They tried to make painting as intuitive as possible and to recreate the actual painting experience. The artist can paint directly on the tabletop display and, thus, is not impeded by the indirection of a mouse or a tablet. To apply paint on this interactive canvas, the artist uses brush-like tools with six degrees-of-freedom. The 2D position of the brush as well as its rotation in x, y, and z are used for painting and the z or height value is used to enable dipping into the paint circles. Color mixing occurs in a mixing area which provides primary colors and mixing circles. Their system encourages creativity and collaborative painting.

(a) Painterly rendering. (b) Decorative mosaic. (c) Pointillism.

Figure 2. Three examples from HAEBERLI's PAINT BY NUMBERS. These images show one of the first attempt to simulate painterly rendering, decorative mosaics, and pointillism.

Modern digital painting systems can be divided into systems that use pixels as the image representation and systems that use a vector description for image primitives. Pixel-based systems (e. g., Adobe® PhotoShop®, Corel® Painter® and TwistedBrush) use a raster array of pixels to store colors in. Vector graphics systems (e. g., Adobe® Illustrator®, CorelDraw®, and Inkscape) instead are using geometrical primitives based on mathematical definitions, such as lines, polygons, textual shapes or groups of such objects, to represent images. There are different graphics representations and applications for different purposes. Unfortunately, both of them yield several disadvantages for the user, when being used separately, as shown in the motivation Section 1.1. The best solution for the user would be to combine the benefits of both representations.

A more extensive overview of digital paint systems can be found in [Smith 2001]. In general, painting systems allow the user to create digital imagery from scratch or with the help of sophisticated algorithms and tools. They usually focus on the user input as the main source for handling the features of the system.

### 2.1.2 *Interaction in Non-Photorealistic Rendering*

With the emergence of NPR, there were several attempts to integrate the user into the act of creating non-photorealistic imagery. Researchers realized that the user input is a vital element of expressive and meaningful paintings.

The interactive system by SALISBURY et al. [1997] is designed to create pen-and-ink-style line drawings from greyscale images. The system provides an editor where the user can change the tone and direction of the image components. Editing tone and direction works similar to a conventional paint program. The user has certain tools to color, increase or decrease the brightness or to invert portions of the image. Since the direction values are represented like an array of pixels, editing the direction works similar to the tone. The system represents a valid effort to include the user's participation in the hatching technique.

Stipple drawings, another NPR interest area, are made interactive with the system FLOATING POINTS by DEUSSEN et al. [2000]. The authors present a fast method to create stipple drawings and they include a collection of tools to alter the drawing. These tools allow the user to manipulate the set of stipple dots after the initial placement. The user needs to be able to alter specific features manually to enhance details or add and delete points. The degrees of freedom in stipple drawings are dot spacing, dot size, dot shape, and inverse drawing. In order to be productive, the user manipulates a large number of points simultaneously. This seems to be a good approach for the effective control of large parts of the image.

KALNINS et al. [2002] presented a What You See Is What You Get (WYSIWYG) system for drawing stroke-based NPR styles directly onto 3D models. WYSIWYG describes a user interface that allows the user to view something very similar to the end result while the image is being created. The system by KALNINS et al. allows the user to control brush and paper style, as well as the placement of individual strokes. In their research, they found that their system and the different styles have a learning curve, but as with a traditional medium, it becomes natural once mastered. It gives a wide degree of control to the users and lets them work in a more direct way with the model.

There are also some alternative forms of user-interaction for NPR. For example, in [DeCarlo and Santella 2002] and [Santella and DeCarlo 2002] an eye-tracking device records the eye-movements of the user to determine where the image should be more abstract. The paint system DAB uses a haptic paintbrush device to allow the user to paint on a virtual canvas [Baxter III et al. 2001].

The goal of these systems is to provide the users with tools that allow them to directly interact with the model to create the image. The WYSIWYG approach is especially useful in reducing the discrepancy between the painting gesture and the results in the image. A computer should help the artist with the tedium or complexity of their work, but not significantly impact their style and preferred ways of thinking and working [Meier 1999]. This work tries to further expand the interaction possibilities and to offer novel and fast painting techniques. One way to allow a better interaction with image elements and details is to use rendering primitives.

### 2.1.3 *Using Rendering Primitives*

As opposed to stroke-based rendering (SBR), where specialized algorithms are used to place discrete elements, this work focuses on allowing one to influence the rendering process. Although discrete primitives are used to model an image, the user is the one to steer and manipulate them. Ideally, a human artist using a system should have total control over the decisions made [Hertzmann 2003].

Using rendering primitives to represent the individual elements of a non-photorealistic rendering is widely explored by MASON et al. [2001a; 2001b; 2004; 2006]. Her SURREAL engine is a multi-agent framework for artistic expression. A picture element is embodied by an agent pursuing its goals. Agents exist in the so called agent-space and render their

appearance into the canvas-space. Agents with the same properties are grouped together to form tribes, and a coalition is formed when an agent discovers that the fulfillment of its goals is compatible with the goals of other agents. The intention of the engine is to lay the control back into the hands of the artist. Another goal was to expand the set of tools for the artist to create expressive images and gain control over algorithms. The SURREAL engine was used to recreate images with styles such as those developed by PIET MONDRIAAN, the Japanese *seigaiha* style, and oil paintings.

SCHLECHTWEG et al. [2005] presented a unified approach for different styles such as stippling, hatching, painterly rendering, and mosaics. Different styles are represented by different kinds of RENDERBOTS, that are all using the same environment and control algorithms. Each agent changes its position in the environment and interacts and communicates with other agents. The RENDERBOTS system produces remarkable results and the set of agents could also be extended to produce further styles. The agents can be controlled to a certain extend once they are created.

The main contributions of these two systems are their sophisticated algorithms for positioning and steering rendering primitives. What they are still missing is a good and accessible user interface. In order to get the users involved and to enable them to actually use a program, one must try to avoid unclear parameters and provide widgets that support fast and easy interaction.

Composing a digital image out of a number of small primitives not only opens up the possibility to interact with them and manipulate their individual parameters, but is also well-suited for such non-photorealistic styles as painterly rendering, decorative mosaics, and pointillism.

### 2.1.4 *Painterly Rendering*

The term painterly rendering comprises all techniques that are trying to render an image with a hand painted appearance (e. g. Figure 3). Most painterly rendering algorithms use brush strokes with a certain shape and size.

The above mentioned system PAINT BY NUMBERS by HAEBERLI tries to produce compositions of brush strokes with a very simple painting model. Strokes are represented as circles, rectangles, lines or scatterings of points and polygons. COHEN et al. [1993] suggest, that drawing an image can also be accomplished by using textured polygons. The brush stroke image may then easily be scaled and rotated. A particle representation for the strokes is used by MEIER in [1996]. Strokes stick on surfaces, to render coherent animations with a painterly style. Her brush strokes consist of a texture image, a size, a position, an orientation, and a color.

In [Hertzmann 1998] an image is composed out of a series of spline brush strokes. The stroke color is matched to the color of a source image. The painting is created fully automatic with a series of layers, that contain differently sized strokes, depending on the detail in the source image. HERTZMANN's framework is able to acquire different visual styles that are described as a set of parameters to the painting algorithm.

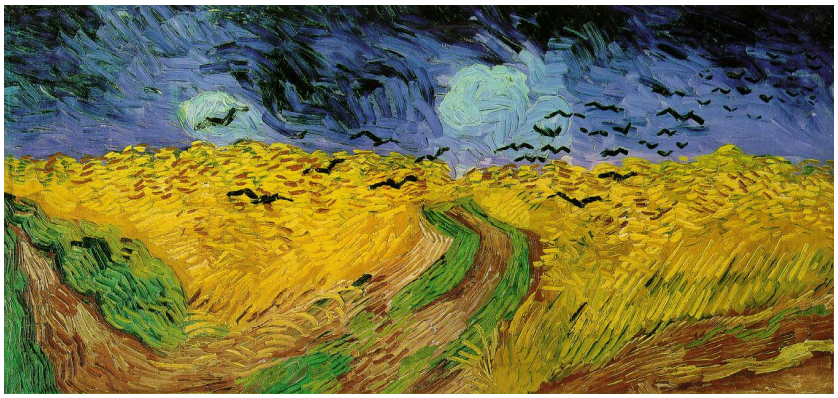


Figure 3. Vincent van Gogh - Wheat Field Under Threatening Skies, 1890, oil on canvas, 50.5 × 100.5 cm, Vincent van Gogh Museum, Amsterdam. The artist uses similar brush strokes throughout the painting using the orientation and the color to differentiate between objects.

In [2001] HERTZMANN improves his algorithm by using an energy function to place brush strokes. The system offers interaction in terms of *metapainting*, where overall styles or different styles for different image regions can be selected. The user can also make suggestions to the algorithm by placing individual strokes.

Research in the area of recreating oil paintings has so far presented mostly automated algorithms that render a painting based on an input image or a 3D model. This work tries to explore the act of image creation from the users point of view where they can interact with brush strokes in novel ways. A user should be able to interfere and use the computer as a supportive tool.

### 2.1.5 *Decorative Mosaics*

Traditional decorative mosaics are images made by cementing together small colored, polygonal tiles made of stone, terracotta, or glass (e. g. Figure 4). These tiles carry information such as position, shape, and orientation and thus provide more information than a pixel image. They are aligned such that edges and other features are emphasized. When being imitated in NPR, this alignment is usually done automatically.

HAUSNER presented a method for creating decorative mosaics via simulation [2001]. Prior to the simulation, the user is able to specify some edge features that should be considered by the algorithm. The tiles are positioned using centroidal voronoi diagrams with a Manhattan metric. The size of a tile is calculated by the algorithm as well and the color is retrieved from a color input image.

ELBER and WOLBERG [2003] present an alternative rendering algorithm where feature curves are detected and extracted from the input picture. A number of offset curves are computed and the tiles are aligned accordingly. The resulting positioning and orientation fulfills the requirement of a decorative mosaic, but they also exclude the user

Figure 4. Irina Charny - Crane, 2001, vitreous glass, 12 × 12 in. Image courtesy and © of Irina Charny. Notice the circle wise orientation of the background tiles. This is rather hard to imitate with today's mosaic algorithms.

and prohibit any creative choices regarding the alignment.

A more complete survey of digital mosaics can be found in [Battiato et al. 2006]. The existing algorithms are well suited for placing and orienting tiles automatically. But once the algorithm has finished, the user does not have the possibility to make changes or to realign portions of the image. Sometimes it is impossible for an algorithm to find a meaningful orientation of the tiles. That is where the user should be able to alter the image and adjoin creative decisions.

### 2.1.6 *Pointillism*

Pointillism is a Neo-impressionistic style where an image is composed out of a large number of small separate points. It was mainly established by Georges-Pierre Seurat (see Figure 5) who uses dashes with an average size of 0.27 – 0.54 cm which would results in a pixel size of 9 – 17 pixels at a screen resolution of 72 dpi [Yang and Yang 2006]. These dimensions have to be taken into account when simulating pointillism

on a primitive basis.

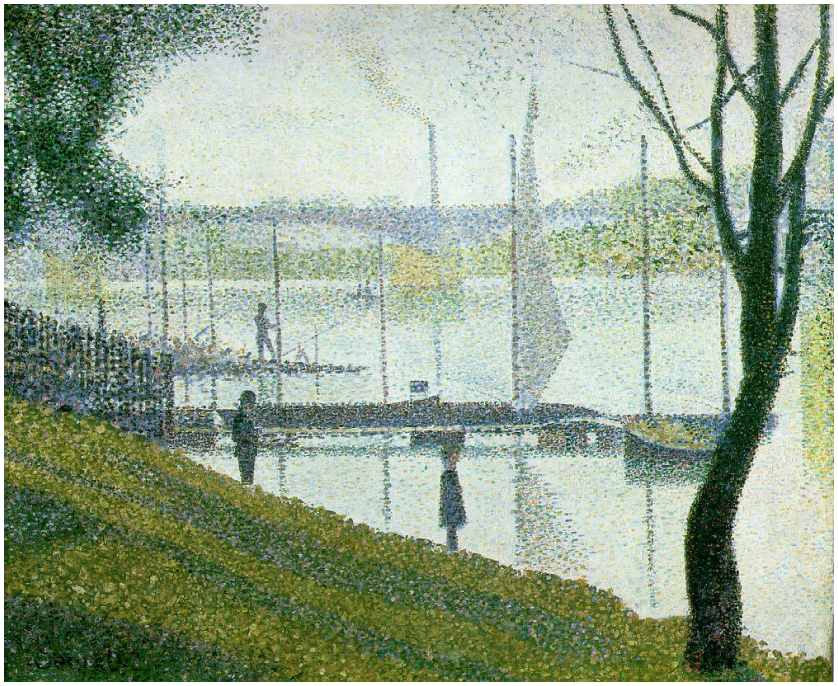


Figure 5. Georges-Pierre Seurat - Le Pont de Courbevoie, 1886-87, oil on canvas, 18 × 21 1/2 in., Courtauld Institute, London. This painting was chosen because it is a very good example for pointillism. It consists of a huge number of dots which have vivid differences in their colors.

HAEBERLI's PAINT BY NUMBERS is the first attempt on creating pointillism styled renditions. He uses circle shaped brush strokes, but presents it as a side product of his application. Although very simple it mimics the effect of a pointillistic painting.

BUCHIN and WALTHER [2003] implemented real-time hatching using programmable graphics hardware. As an application they demonstrated a pointillistic rendering style by choosing a color for each stroke depending on the background color. One conclusion that can be drawn from their work is, that neighboring points must vary in their color to achieve the pointillism effect.

The authors of [Fischer and Bartz 2005] are using a non-photorealistic pointillism style for augmented reality. To reduce the difference between the appearance of virtual and real objects, both objects are displayed with a pointillism look. All real objects from the video stream are filtered with special non-photorealistic image filters and all virtual objects are rendered and then stylized in a similar, pointillism way. In their application, each brush stroke, which represents a pointillism primitive, has a position, a radius and a color offset, which is later on added to the original color value. This work makes clear that pointillism strokes are not only simple dots, but primitives with a number of parameters influencing the final appearance of the painting.

YANG and YANG [2006] analyzed and tried to mimic paintings by

Georges-Pierre Seurat using small image elements as the base for their algorithm. The shapes of the dots vary from circles to ellipses and are randomly selected in the generation process. The primitives are represented by scanned brush dips and they are oriented based on image gradients. The gradient underneath the center pixel of a dot is calculated and the dot is rotated such that the longer axis of the ellipse is perpendicular to the gradient direction. Like Seurat, their algorithm is only using a palette of 11 colors, mixtures of these colors with white and pure white.

In summary these related works have shown that the primitives that yield pointillism-like paintings are shaped like circles, ellipses or similar forms. Attributes that users can change are their size, color, orientation, and possibly their shape.

## 2.2 INTERACTION ON LARGE DISPLAYS

To provide an intuitive way of interacting with the painting, transferring a painting system onto a very large, high-resolution displays seems to be a worthwhile objective. These displays allow users to work up close with detailed information and also enable them to step back and look at an image in a large scale.

Tan et al. [2003] present the results of two user studies where they compare the performance of users on a large display to users working on a standard desktop monitor. Their study demonstrates that users generally benefit from working on a tabletop display, because they become more immersed in the task on a large high-resolution display. To further study the user performance on large interactive displays, Tyndiuk et al. [2005] perform experiments on different kinds of 3D interaction tasks when using either a standard desktop display or a large immersive display. They show that manipulation tasks can benefit from a visualization on large displays.

Apitz and Guimbretière [2004] developed CrossY, which uses crossing to overcome the difficulties of using standard WIMP-interfaces (Windows, Icons, Menus, and Pointers) on tablet PCs with direct pen interaction. They confirmed that pen on a tablet is a rather noisy input device. This has to be taken into account when using a pen or a finger on a tabletop surface as well. Guimbretière [2000] also presented FlowMenus as a kind of marking menu that was developed for the use on large, high-resolution displays. These interaction metaphors try to overcome difficulties common to the interaction with large, interactive displays.

Painting on a large, high-resolution display can improve the painting experience. They allow the direct interaction with image primitives and viewing them in high detail. The following sections are going to analyze the advantages of working with large, high-resolution displays and the fundamentals of an appropriate user interface for these displays. Interactive displays are not steered with mouse and keyboard, instead a finger or the hand is used. Therefore new interaction metaphors have to be used that differ from the paradigms of the desktop world.

### 2.2.1 *Direct Manipulation*

To reduce the indirection between the user input and the actual effect on the screen, an interactive touch display can be used. There are three main categories of interaction styles with a computer [Newman and Lamming 1995]:

- *key-modal*,
- *linguistic*, and
- *direct manipulation*

Key-modal interaction describes user interfaces that are operated mainly by using function keys or a keyboard. It has a number of different modes which represent certain behaviors. Common key-modal interaction styles are menu-based interaction, question-and-answer, function-key interaction, and voice-based interaction. A linguistic style means, that all of the users input is done with an alphanumeric keyboard, using a particular set of conventions or a special language. Examples for linguistic styles are command-line interaction and text-based natural language. Direct manipulation describes systems with the following qualities [Preece 1994]:

- visibility of the objects of interest,
- rapid, reversible, incremental actions, and
- replacement of complex command language syntax by direct manipulation of the object of interest.

There are two widely used styles of direct manipulation. Graphical direct manipulation and form fill-in. Form fill-in aims at non-expert users and is used to allow simple data entry into pre-defined form fields. Graphical direct manipulation means that information is displayed in the form of graphical objects, which can be manipulated graphically with a pointing device. After each user action, the display of the objects concerned, responds immediately to show any resultant change.

Well-designed direct manipulation systems may elicit enjoyment from their users, because novices can learn the basic functionality quickly and the users can immediately see if their action are leading to their goals.

### 2.2.2 *Bimanual Input*

Having the capability of processing more than one input at a time, allows the integration of bimanual input to decrease the disruptiveness of using the interface while painting. Guiard [1987] divided every-day activities into three categories of how we distribute the work between our two hands. Activities such as combing one's hair are *unimanual*. On the other hand, there are activities called *bimanual*, when using both hands. Weight lifting for example is a symmetric bimanual task, whereas playing a stringed musical instrument, for example, is an asymmetric bimanual one.

Bier et al. [1993] demonstrated the use of asymmetric bimanual input with their Toolglasses and Magic Lenses. The user is able to position a palette over the object of interest with one hand and browse through the palette with the other hand. They demonstrate the successful use of an interface for two-handed input and mentioned that could also be applied to other screen-based applications, e. g. paint programs.

In their study of hand shape use in tabletop gesture interaction, Epps et al. [2006] suggest the use of both hands to improve the naturalness and perhaps the speed of various manipulation tasks. Two-handed input is well suited for high-degree-of-freedom symmetric interactions but it can also be useful in asymmetric interaction tasks. In the real world, we frequently use both hands to perform certain tasks [Moscovich and Hughes 2006].

In the study by Lank et al. [2006] about techniques for non-preferred hand mode switching in stylus interfaces, they discover that concurrent bimanual user input is faster than having the mode switching before or after the drawing gesture. Switching modes usually causes cognitive and motor costs. Therefore, the goal must be to reduce the overhead of switching modes. Using two-handed input means, that the non-preferred hand does the mode switching and the preferred hand performs the actual gestures. In their study the users rarely used actual parallel interaction, but it was easier and faster to leave the non-preferred hand at the mode switching tool, instead of interrupting the preferred hand from gesturing. The study also reveals, that the action of the non-preferred hand does not interfere with the action of the preferred hand.

Bimanual input is also common in video games. The vast majority of game control devices are designed to receive some input from both hands [Morgan et al. 2006]. Users can generally benefit from the feature of bimanual input in a painting program in that they are able to manipulate the user interface with their non-dominant hand and paint as usual with their dominant hand. This is very similar to holding a palette in one hand and using a brush with the other one.

### 2.2.3 *Click-less Interface*

As already mentioned in Section 2.2 the standard metaphors from the WIMP world are rather unsuitable for the interaction on large, interactive displays.

Apitz et al. [2004] noted that while the current interfaces were designed in an indirect pointing configuration with a stable pointer controller, tablet computing provides a direct setting with a pen, but a rather noisy input. The same holds for large interactive displays using the finger as the direct input device.

An interactive display does not necessarily support inputs like double click or right mouse click. Furthermore, activating a tool by choosing an entry out of a linear menu is not very suitable for an imprecise input device such as the finger. A famous alternative to linear menus are *pie menus*. Pie menus are two-dimensional, circular, and in many ways easier to use and faster than conventional linear menus [Hopkins 1991]. Usually the menu item target regions are shaped like the slices of a

pie. The relative direction of the pointing device is used to determine the selection. The center of the pie is usually inactive, so releasing the pointing device, namely the finger, without moving, dismisses the menu.

Pie menus work well with alternative pointing devices just like they can be found on interactive displays. Because a pie menu is a self-revealing gestural interface, it is easy for novices. In order to prevent them from getting too big, icons should be used instead of labels.

## 2.3 COMPREHENSIBLE USER INTERFACES

Relatively little emphasis has been placed on the problem of how to provide direct, intuitive interfaces that users can use to access NPR algorithms or to make changes in the image. MEIER [1999] stated that an effective user interface is essential for an application to be used by an artist.

A user interface is build up by combining a number of *widgets*. In graphical user interfaces, a widget is a combination of a graphic symbol and some program code to perform a specific function[1]. Another important aspect of the user interface is the *Look and Feel* [Olsen Jr. 1998]. The look of a widget is its visual appearance and the Feel is determined by how it reacts. The look and feel is important for the following three issues:

- Ease of learning: A widget has to be easily understood by the user.
- Ease of use: A widget has to be easily accessible and effective.
- Attractiveness: A widget has to be attractive and nice-looking because the users must like the tools they are working with.

Arranging interface elements, such as widgets, in a user interface is not a trivial task to do. They must be regularized on many levels simultaneously to establish a pattern for the viewer [Mullet and Sano 1995]. According to MULLET and SANO there are some general strategies as well as specific guidelines for the positioning, grouping, and relationship of interface elements. Some general strategies used are:

1. Use regular geometric forms, simplified contours, and muted colors wherever possible.
2. If multiple similar forms are required, make them identical, if possible, in size, shape, color, texture, line weight, orientation, alignment or spacing.
3. To reap the benefits of regularity, make sure critical elements intended to stand out in the display are not regularized.

Specific design guidelines for the development of a user interface are [Mullet and Sano 1995]:

1 From FOLDOC, the Free Online Dictionary of Computing, http://foldoc.org, last visited: 20.02.2007

POSITION: Positional alignment of widgets reduces the complexity of an interface by making the global form cleaner and more understandable.

GROUPING: This is used to bind functional units tightly together while distinguishing them from the surrounding controls. Widgets that belong to a group of similar functionality should have a high degree of similarity in terms of shape, size, color and surface texture.

RELATIONSHIP: The relationship between elements is used to imply the meaning of each element by its location relative to its surrounding elements, so that the need for explicit labeling is reduced.

To represent the functionality of a widget, e. g. a button, one can either use text-labels or *icons*. Text-labels are usually unambiguous whereas icons are language-independent and easy to be recognized once they have been learned [Horton and William 1994]. An icon represents a widget and, therefore, a capability of the system. Icons with a distinctive shape are easy to find. For icons to be effective, they must be simple and easily discriminable [Byrne 1993]. To keep a user interface as small as possible, icons are preferred as well, because a well-designed icon says much more in fewer pixels than a label does.

An important part of the visual feedback of any widget is to indicate whether it is active or inactive [Olsen Jr. 1998]. Users have to know if they are performing the right input. If, for example, a button is pressed, it should visually show, that its status has changed. This is also necessary for an intuitive manipulation on large, interactive displays, to differentiate between modes.

## 2.4 SUMMARY

Painting systems are, nowadays, common applications to create digital imagery. They incorporate a vast amount of features but usually suffer from complex interfaces that are crowded with parameters. To use an NPR algorithm the users are generally left on their own, using trail and error to ascertain what the programmer meant by certain parameters. In general, the accessibility of NPR algorithms and the possibility to interact with them is still to be improved. One way to gain a more direct interaction with the image and its generation process is to use rendering primitives which are going to be explained in detail in the following chapter.

The painting experience can also be improved by applying a painting system onto a large, high-resolution display. These interactive displays offer direct manipulation, which allows the user to directly touch the canvas with the finger or a pen and control visualization components. An interactive display with multiple inputs also affords bimanual input. This enriched way of interacting with a painting system allows the user to work faster with less interruptions from moving a hand from the interface to the canvas. Because of direct manipulation using a finger, a different set of interactions metaphors have to be considered to facilitate adequate interaction.

# 3 CONCEPT

This chapter presents conceptual details of this thesis. The discussion shall clarify how the creation of a non-photorealistic rendering differs from previous approaches and emphasize important aspects of the applied technique. These include how to re-introduce interactivity into the image creation process and how to ensure a responsive behavior of the system. It also addresses the underlying concept of the system, in particular, how components work together to make the rendering interactive.

## 3.1 AN INTERACTIVE IMAGE CREATION PROCESS

There have been different kinds of efforts to incorporate interactivity into the process of creating non-photorealistic imagery. Usually an algorithm is set up in a pre-processing step or a filter for NPR stylizations is applied in a post-processing step. Some systems then allow the users to make suggestions to the algorithm or even steer it in a certain manner. Interactivity could also appear in a way that attributes are specified for the stylization, e. g. direction fields or example strokes. An even more interactive approach is to directly process and integrate the users' input. Some recent implementations allow to directly paint on models or to use real paintbrushes instrumented with trackers to provide direct interaction. In spite of that, very little effort has been made to introduce interactivity into the image creation process in its entirety and to apply local changes in a flexible manner.

Usually, the rendition of such an image requires the steps depicted in Figure 6. The user provides some kind of an input model by specifying a geometric model and a set of parameters. The model is one or more of these types of sources for non-photorealistic renderings:

- 3D surface models, e. g. [Kalnins et al. 2002; Salisbury et al. 1997; Vanderhaeghe et al. 2006], such as polygonal meshes or parametric surfaces;
- 3D volume models, e. g. [Ebert and Rheingans 2000; Burns et al. 2005];
- Implicit models using iso surfaces e. g. [Foster et al. 2005; Wyvill et al. 2005];
- Point clouds, e. g. [Xu et al. 2004; Xu and Chen 2004];
- An image, e. g. [Deussen et al. 2000; Schlechtweg et al. 2005; Park and Yoon 2004; Hertzmann 1998; Curtis et al. 1997], which can be a digital photo, a scanned painting, or even G-buffers; or
- User input, e. g. [Mason 2006; Lank et al. 2006].

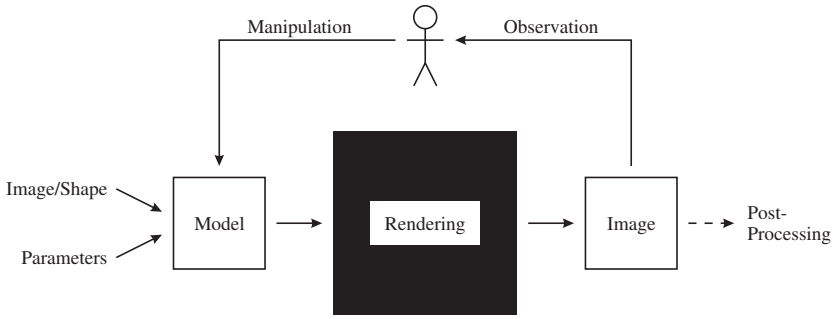



Figure 6. Traditional non-photorealistic image creation process. The user inputs a model which is then processed and rendered into an output image. The rendering algorithm behaves like a black box in this case.

The amount of parameters is usually rather large and requires trial and error to find the right combination. The model is then processed and rendered into an image. Some rendering algorithms also have a post-processing step to apply image filters or create additional effects. If the result is not satisfying, one has to go back and make changes to the model and/or tweak the parameters. The image components, such as brush strokes, are usually fixed in the result image and hard to manipulate. Then the rendering has to create a new image. A good example for these steps can be found in the "ant" approach by Semet and Dur [2004]. The only control the user has is to make changes to the model, including the parameters, and then observe if they result in the intended changes in the output image. The rendering itself remains a black box for the user. The actual program being executed can not be examined by the user. A black box algorithm also means, that the user cannot see its inner workings.

To address this problem, this system uses a rendering process called *modeling with rendering primitives* [Mason 2006] (see Figure 7). This approach tries to bring the model creation step and the actual rendering closer. The user is constantly interacting with rendering primitives that are being rendered instantaneously. The model is integrated into the

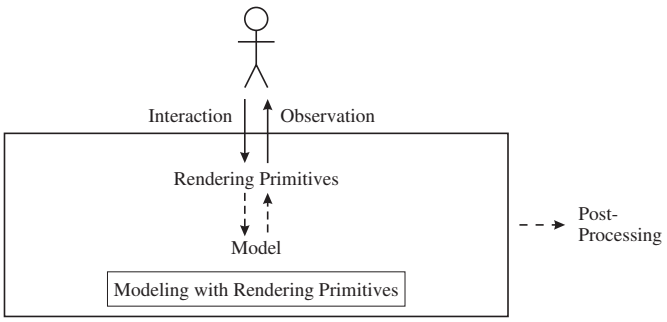



Figure 7. A novel interactive image creation process.

rendering process and is constantly being changed. The user observes the current status of the rendering and applies local changes where desired. The model is build and adjusted throughout the creation process. The main advantage of "modeling with rendering primitives" is that the users work with the image components and immediately see the result of their actions. This allows for experimenting and making even very small changes without the immoderate overhead of choosing parameters or re-applying an entire algorithm.

## 3.2 RESPONSIVE INTERACTION USING BUFFERS

The system is build upon the UNIVERSITY OF CALGARY TABLETOP FRAMEWORK which provides reusable and extensible interfaces on which to build applications [Miede 2006]. It also makes use of buffers to regain responsive interaction on large, high-resolution displays [Isenberg et al. 2006]. Buffers are two, or more than two, dimensional arrays that store different kinds of properties. One advantage of buffers is their fast access to data and the local awareness by mapping a position to a certain buffer value.

Applications using the framework are comprised of *visualization objects* and *interface components* (see Figure 8). A visualization object carries information to be communicated and is the visual content that the user sees. An interface component is responsible for controlling and monitoring visualization objects and is manipulated by user interaction.

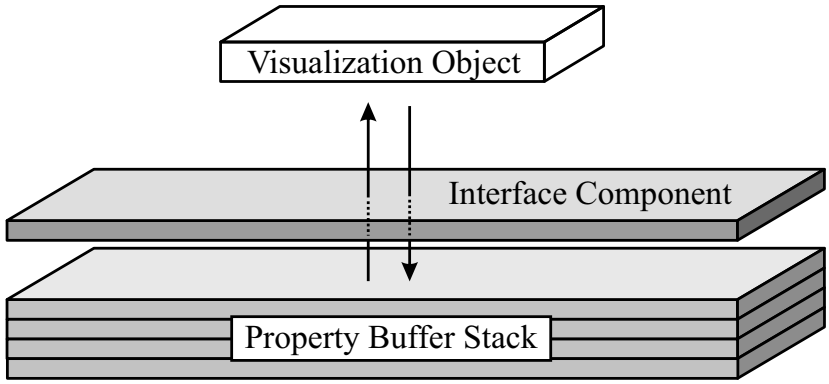
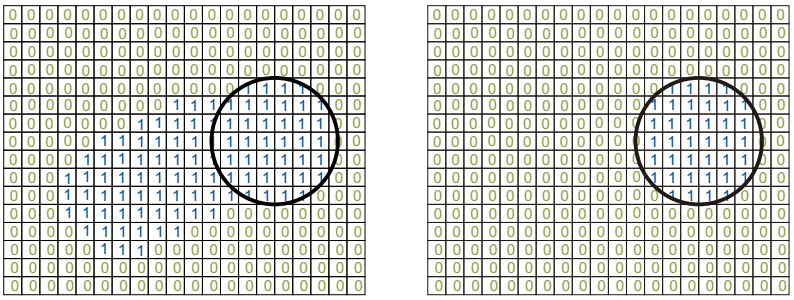



Figure 8. Indirect communication between visualization objects and interface components. Visualization objects are accessing the property buffer stack provided by the interface component.

The monitoring and steering of visualization objects can be a very costly task, especially with a large number of objects. To reduce communicational and computational effort, buffers are used to replace the communication between interface components and visualization objects. These buffers store properties of interface components and make these properties available for visualization objects. Buffer values are only changed if a system component computes new values and writes them into the buffers. By this, expensive calculations are only necessary

when modifying buffer values in a certain area. Visualization objects only interact with the saved buffer values at their position and do not need to communicate with the interface components directly. These *property buffers* store scalar values or vectors of two or more dimensions. If more than one value is used by an interface component, a *buffer stack* is needed.

Buffers can be divided into *persistent* and *instantaneous* buffers. A persistent buffer stores the actual properties of visualization objects, whereas instantaneous buffers only store the changes to properties. That means a primitive always needs to know properties like its size, color, or orientation. Therefore, persistent buffers are used that keep their entries until changed. The content only changes if an interface component manipulates them (see Figure 9 (a)). The content in an instantaneous buffer, on the other hand, only exists for one render step and is being reseted to a default value afterwards (see Figure 9 (b)). An example application for such a volatile, instantaneous buffer could be motion information. If a tool writes movement vectors into the buffers, in order to affect the position of primitives, the buffer has to be reseted in the next step to avoid that the primitives keep on moving.



(a) Persistent buffer. (b) Instantaneous buffer.

Figure 9. Instantaneous and persistent buffers. Top view on the two different kinds of buffers with a tool writing into them. The values in a persistent buffer stay changed after a tool writes into it, an instantaneous buffer is set to default values.

## 3.3 SYSTEM DESIGN

The system described in this thesis is trying to imitate aspects of real painting. The user paints on a background surface which represents the canvas. The canvas in this system is an interface component that holds the buffer stack. Some example types in this property buffer stack are:

- Size (scalar)
- Speed (scalar)
- Erase probability (scalar)
- Orientation (2D vector)

- Motion (2D vector)
- Position (2D vector)
- Color (3D vector)

The buffer stack also contains a special buffer, the *parameter-buffer*, which holds the settings for the tools as well as the current system state. This buffer is much smaller than the property buffers, which span over the width and height of the output display. It is used to facilitate a fluid communication between the system's components (see Figure 10). The parameter-buffer, like the property buffers, is maintained by the canvas and is being initialized with default value by it.

The rendering primitives, which were introduced in Section 3.1, are the visualization objects in this system. They reside on the canvas and read their properties, such as size, orientation, or color, from the property buffer stack. Primitives might access the parameter-buffer to to get information about the system state.

To interact with the rendering primitives, the the system provides a tool that allows to alter the buffer stack. This tool is using the parameter-buffer in order to know what kind of values it has to write into the property buffers.

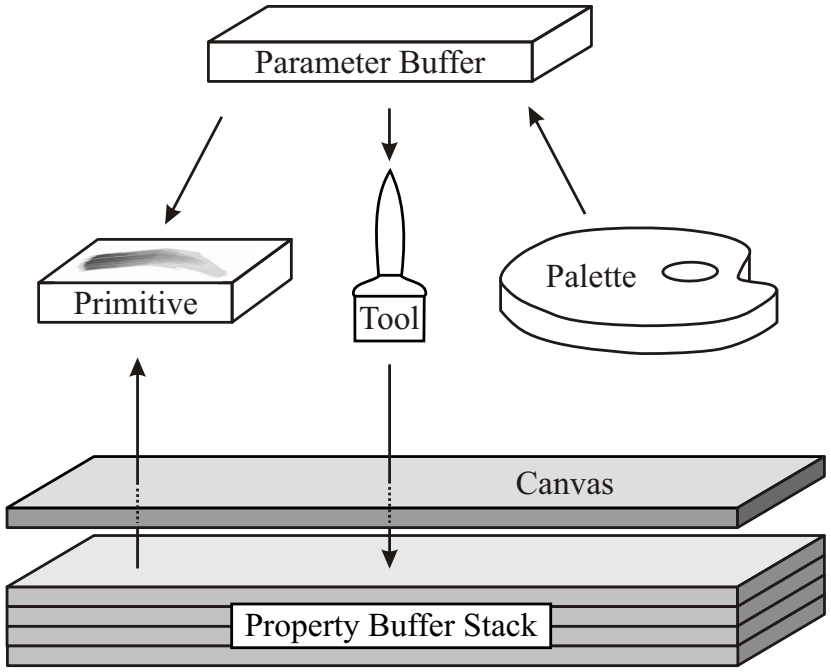



Figure 10. Communication between system components. A parameter-buffer is used to enable a fluid communication between the main components.

Another important part is the user interface component. As an analogy to real painting it is designed like a palette. The palette holds the means for the user to change a mode or select a property for a tool.

These changes are then written into the parameter-buffer to make them available for other system components.

## 3.4 APPLIED CONCEPTS

The system, described in detail in the next chapter, is based on the concept of "modeling with rendering primitives". Entities that have the appearance of pixel graphics and a number of the flexibilities of vector graphics provide a nice visual character and also an uncomplicated way of manipulating them. A stack of property buffers is used to steer them and change their attributes. This buffer concept enables a responsive interaction with the primitives and is also used to manage the communication between the major system components.

# 4 IMPLEMENTATION

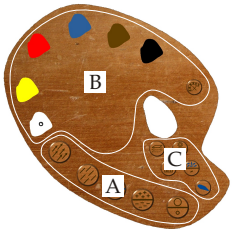
After having introduced the main concepts for the interaction with painting and the application of buffers, this chapter addresses the realization, implementation and usage of the proposed system. In particular the user interface, the different tools, and implementation details for the main system components are discussed.

## 4.1 USER INTERFACE

The user interface is represented by the widget in Figure 11, which is designed like a palette to imitate a similar experience as in traditional painting. The palette can be moved on large displays by the user, with just one finger, as it supports Rotate'N Translate (RNT) [Kruger et al. 2005]. As suggested by KRUGER et al. the palette supports a translate-only region in its center. If the user touches the peripheral RNT area, the palette behaves like an object moving against friction and orients itself like a sheet of paper dragged on a table. This feature can be used to ease reading and viewing issues known to large tabletop displays [Stacey D. Scott and Habelski 2005].

The palette can also be tossed across the table to ease reaching issues on large screens. This is particular helpful if more than one

(a) The palette.




(A) Contains the main buttons for the tools and their options.
(B) The mixing area consists of a number of colors to mix with and an empty wooden part to mix on.
(C) Additional buttons to steer system functionalities.

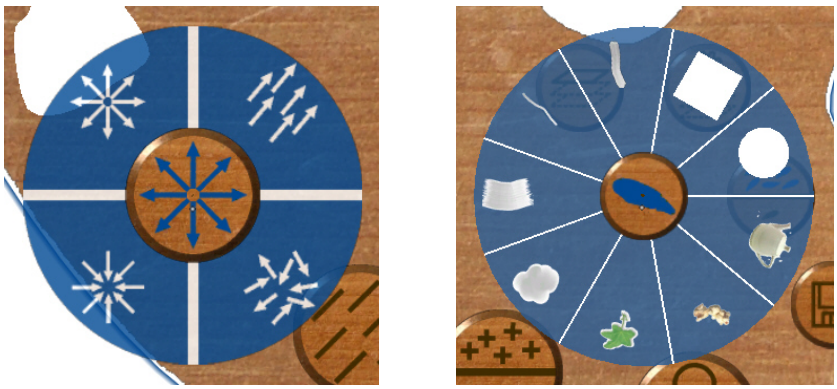
(b) The different areas on the palette.

Figure 11. The palette. It represents the user interface and holds widgets to visually control the system state.

user is working on a painting and somebody wants to give another person, at the other end of the table, the opportunity to contribute to the painting. If multiple users want to work simultaneously, multiple identical palettes can be created. In this case, changes made on one palette would affect the same parameter-buffer within the system and thereby change the settings on all the other palettes as well. How this would influence collaborative painting and if this behavior is even appropriate, might be a question for future work.

LAYOUT DECISIONS: The palette is divided into three areas. In the bottom part A (see Figure 11 (b)) one can find all the different tools. The buttons for the tools are of equal size and shape, and they are aligned in a row. The mixing area can be found in area B (see Figure 11 (b)). This area can be used to mix colors or to pick one of the initial *color-swatches*[1] from the perimeter of the palette. color-swatches are arranged according to the color blots on a real palette to imply the meaning of these interface elements (see Section 2.3). In part C (see Figure 11 (b)) of the palette, additional buttons are arranged to represent additional functionality such as layers, choosing the primitive type or taking a screenshot. The buttons in this part are smaller and arranged in a different way than the tool buttons. This makes them easily distinguishable and sets them apart from the tool buttons. Throughout the palette, as explained in Section 2.3, grouping is used to associate buttons with similar functionality. To indicate that a tool is active or a mode has changed, the buttons change their appearance by having the icons appear in another color.

CLICK-LESS INTERFACE: On the palette, users can chose between tools and the type of a certain tool. For this purpose, pie menus (see Section 2.2.3) are used to select a tool or another program function (see Figure 12). A pie menu is presented, if the button for the tool offers different options to chose from. If a tool requires the users to specify a



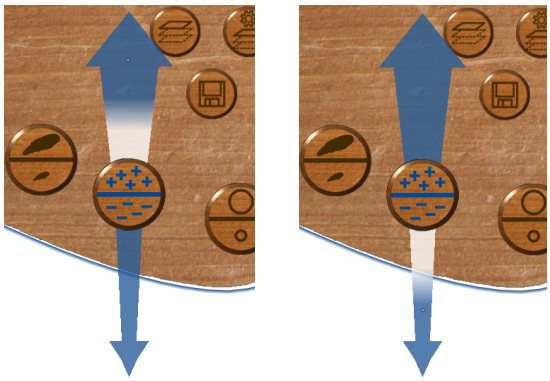
(a) Choosing a motion mode. (b) Picking a primitive type.

Figure 12. Example pie menus on the palette.

1 Color-swatch: A small amount of color used to mix with other colors.

scalar value as its parameter, they can simply touch the button for the respective tool and drag their finger up or down to increase or decrease the parameter (see Figure 13). With this *slider* widget, users can write an integer or a floating point value for a tool into the parameter-buffer.

AVOIDING PARAMETERS: An important goal was to avoid incomprehensible numbers and complicated parameters in order to make the user interface more intuitive. Aiming at the usage on large, interactive displays requires to go away from input paradigms such as a mouse or a keyboard. For example, with the sliders (see Figure 13), the users do not see the exact number they are about to chose, but they can recognize the approximate number because of the visual feedback. A gradient appears on the slider, which does not force the specification of exact values. Most of the time accurate numbers are not even requisite and real painting is all about trying and approximating. This also allows mode change and parameter selection at once with just a single gesture. Visual feedback is given where appropriate throughout the user interface. For example, changing the size of the tool is interactively shown on the canvas. However, no numeric values or text labels are shown.



(a) Dragging up - increasing.

(b) Dragging down - decreasing.

Figure 13. Changing the parameter for a tool.

BIMANUAL INPUT: Interaction techniques such as pie menus and sliders are especially useful when working with the interface on large interactive displays (e. g. tabletop displays, wall displays). Such displays usually afford multiply inputs and, therefore, allow bimanual input (see Section 2.2.2). This allows the users to steer the palette with the non-preferred hand and paint on the canvas with the preferred hand, as can be seen in Figure 14. Of course, this usage of dominant and non-dominant hand would also work the other way round. Bimanual input enables the user to leave one hand where the palette is and use the other where changes in the image have to be made. Having the feature of multiple inputs allows one to change tool parameters and

paint simultaneously. One can imagine to increase the brush size while painting along with the brush, which would not be possible with a real brush.


Figure 14. Using bimanual input during painting.

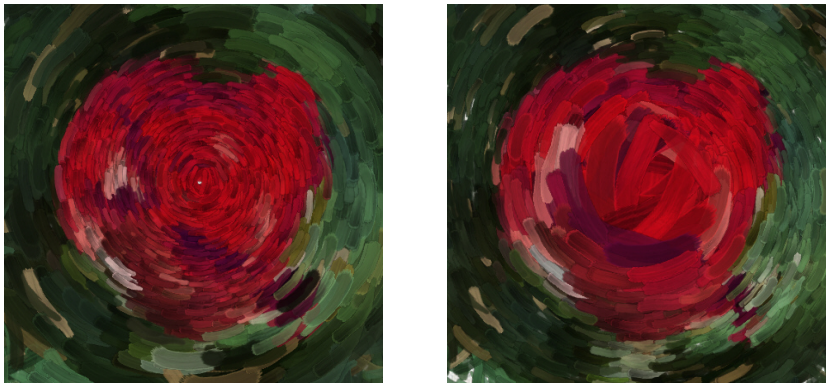
## 4.2 TOOLS

This section contains particulars about how the interface elements work, what the tools do, and how the user applies them. When performed on a large interactive display, painting with this system allows the use of graphical direct manipulation (see Section 2.2.1). The character of, for example, brush strokes is determined by such elements as the color, the direction, the size, and the shape [Park and Yoon 2006]. As analyzed in the motivation in Section 1.1, all of these characteristics can nowadays be calculated automatically such as by Park and Yoon [2004]. This work, however, tries to focus on the freedom of the user and provides tools to interactively manipulate these attributes and to adjust and perfect every part of the painting.

There are two ways of affecting the primitives. For some tools, the user can set a parameter and write it into a buffer. Other tools allow the users to write an input characteristic, for example the motion, into a buffer and thereby manipulate primitives. Reasonable effort has been put into the design of comprehensible and recognizable icons. Therefore, each of the following tool sections includes the icon belonging to that tool.

### 4.2.1 *Size-Tool*

To use strokes of the same size for each region would "flatten" the painting [Hertzmann 1998]. That is why the artist must be able to increase or decrease the size of primitives. In this system the user can change the size of entities with the size-tool (Figure 15). With the slider attached to the size-tool button, the user can specify the size magnification or miniaturization rate.

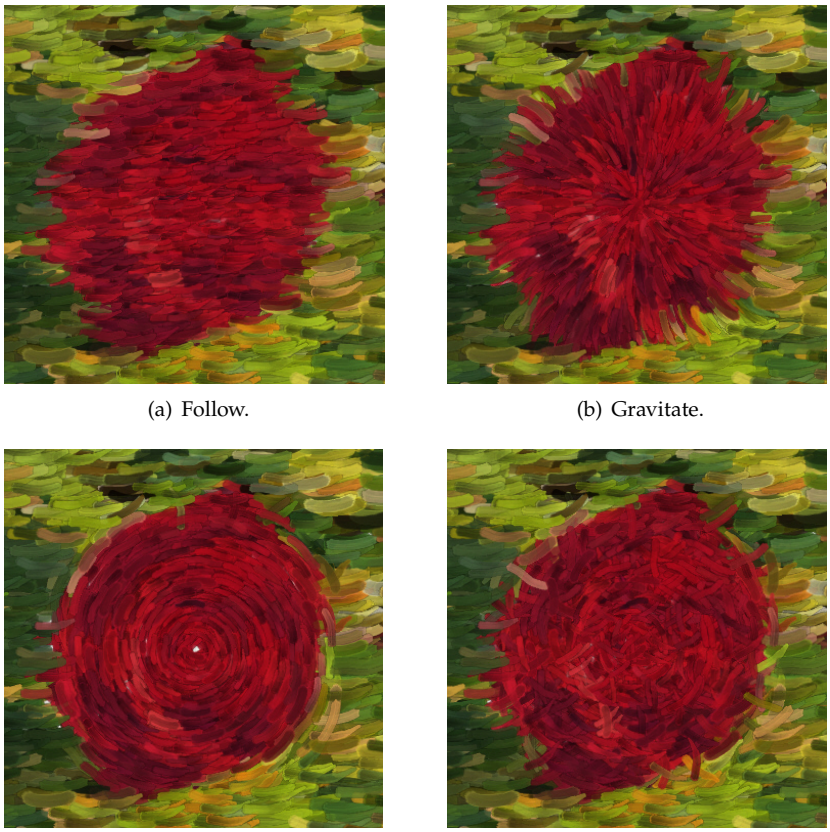


(a) After increasing the size of strokes. (b) After shrinking strokes.

Figure 15. Increasing and decreasing the size of primitives with the size-tool.

### 4.2.2 *Orientation-Tool*

The orientation-tool allows a user to align and rotate primitives the way they want. Different orientation styles have been implemented, which are inspired by Mason et al. [2006]. The *follow* orientation simply aligns the primitives along the user input direction (see Figure 16 (a)). If, for example, the users want to have mosaic tiles representing water to be aligned in a wavy curve, they simply paint a wave with this tool in this area. With *gravitate* the primitives are oriented towards the center of the tool and take up a star-like formation (see Figure 16 (b)). The *orbit* orientation makes the primitives align to circles which are centered around the center of the tool (see Figure 16 (c)). And finally, the *random* type of orientation introduces noise into the stored orientations and basically lets the artist go back to the initial random state (see Figure 16 (d)). The orientation-tool writes the two components of a two-dimensional orientation vector into the persistent orientation buffer.



(a) Follow. (b) Gravitate.

(c) Orbit. (d) Random.

Figure 16. The orientation-tool with different orientation types.

### 4.2.3 *Motion-Tool*

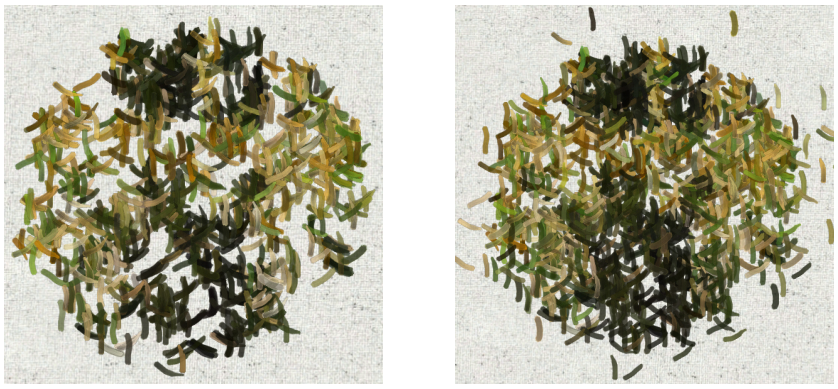
With the motion-tool the user is able to move primitives on the canvas in different ways. If the users want to increase the density of strokes, for example, in a certain place, they can pick up the motion-tool and move them towards this location. As with the orientation-tool, there are different types of motions available. The primitives can be attracted towards the center of the tool (Figure 17 (b)) or they can be pushed away from the center (Figure 17 (c)). It is also possible to make the primitives follow the tool, to position them in a different location (Figure 17 (a)), or to apply random motions. The motion-tool writes into the instantaneous motion-buffer.

(a) Follow. (b) Attraction. (c) Repulsion.

Figure 17. Results after applying the different motion-tool types.

### 4.2.4 *Create-Erase-Tool*

An essential functionality is to add or remove primitives on the canvas. The user can increase the density of strokes or pointillism smudges by creating new ones. The number of primitives created per click can be controlled with the sliders described in Section 2.3. It is possible to create a large amount of new elements at once or even just a single one. For the create-tool, the user can select the type of primitive to be created (see Figure 18). The positions of the new primitives are random within the tool circle.

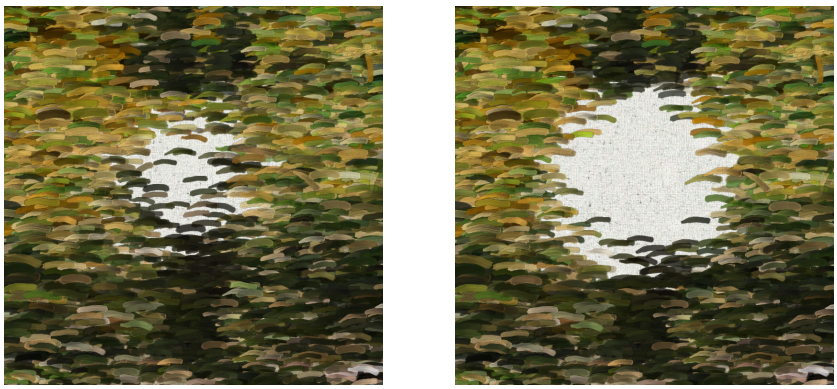


(a) After adding a couple of primitives. (b) After adding lots of primitives.

Figure 18. The create-tool.

After demonstrating the system to artists, one of them said that "an artist must also be able to go back inside a system in order to feel comfortable working with it". This means that, after performing a certain action, there has to be the possibility to undo this action. In traditional painting this can be done by scraping off color from the

canvas. Removing primitives, such as stroke, within this system is realized with the erase-tool (see Figure 19). The users can literally scrap a hole into the painting or they can slowly delete small amounts of strokes from the touched area.



(a) After removing a couple of primitives.

(b) After removing lots of primitives.

Figure 19. The erase-tool.

With the erase-tool the user writes probabilities into the instantaneous erase-buffer. If a primitive recognizes an erase probability higher than zero at its position, it generates a random number. If the random number is smaller than the probability, it deletes itself (see Figure 20). This rather complicated evaluation was chosen to avoid patterns when erasing. The user does not have to delete everything at once at the touched location, which would result in a white spot. This evaluation also enables slow or fast erasing of primitives.

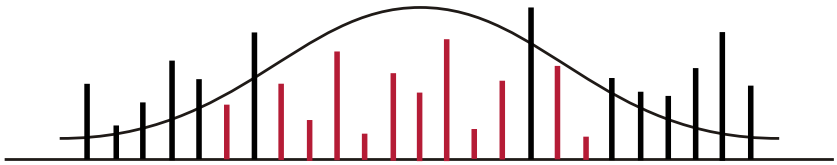


Figure 20. Evaluation of the erase-buffer entries. The curve represents the erase probabilities written into the erase-buffer (1D section). Each bar represents a random number, generated by a primitive. If the random number is smaller than the probability in the erase-buffer at the current position, the primitive gets deleted.

### 4.2.5 *Color-Tool*

The color-tool is embedded within the mixing area on the palette. Users can either choose an initial color-swatch (e. g., red, yellow, blue), which

are placed around the mixing area, or they can create a new color-swatch by mixing two existing ones. The creation of color mixtures is inspired by MEIER et al. [2004]. Mixing two colors works as follows: The users dip their finger, or table-pen if they are using one, into an initial color-swatch and drag it into the middle of the palette. This produces a new color-swatch with the same color at the position the finger was dragged to. In the next step they take another color-swatch in the same way and drop it onto the previously created blot. This mixes the two colors and, in addition, *color gradients* between the new color and the initial colors are generated (see Figure 21). The color gradients provide every color in between two colors and thus offer any mixing ratio of the two initial colors. The layout of the mixing area can be adapted to the users preference by rearranging the blots in the mixing area.

(a) Mixing of primary colors. (b) Mixing with black and white.

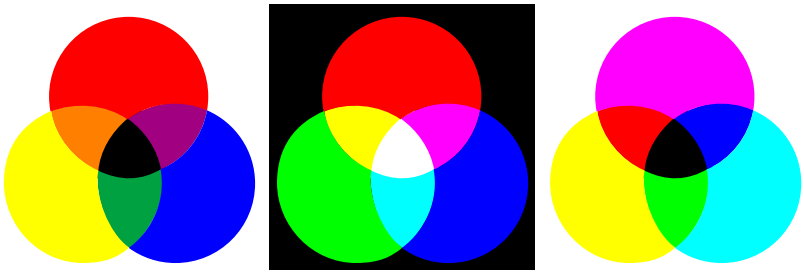
Figure 21. Mixing examples.

This mixing behavior was chosen to recreate the procedure of mixing on a real palette. It was extended with the color gradients and the ability to manage the arrangement of the color blots. The last chosen color is denoted by a small circle in the mixing area (see Figure 21). In addition, the circle shaped representation of the tool takes up the color which is currently in use.

Color mixing works entirely in the Red Yellow Blue (RYB) color space. See Figure 22 for a comparison with other color models. RYB is a set of subtractive primary colors. It is used by artists and in art education. Every color is internally represented by an RYB value. The colors are only converted into RGB if something is to be rendered with OpenGL.

The RYB model is not able to produce several bright shades of green, cyan, and magenta, but there are several reasons why RYB color model is still used in this system. Due to the use of a subtractive color model based on pigment mixing in early childhood art training, many people have a mental model of color that is quite different than the RGB or CMYK model [Gossett and Chen 2004]. The RYB color theory has become heavily used by all kinds of artists, and has thus been heavily established as the norm despite its inaccuracies, which are generally either overlooked because of its popularity or simply unnoticed. White and

black have been added to the palette to simplify the access to darker or lighter colors.

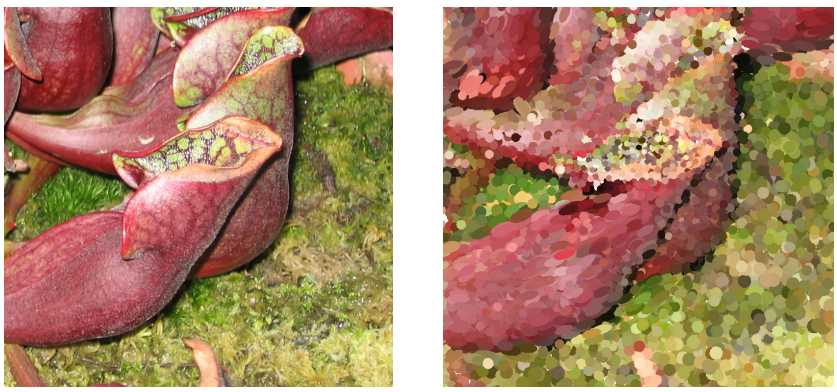


(a) RYB. (b) RGB. (c) CMYK.

Figure 22. Mixing behavior in different color spaces.

### 4.2.6 *Additional Tools*

There are also some auxiliary functions available through the user interface. One of them is the possibility to load an image file and copy its color components into the color-buffer. Sometimes a user finds it difficult to start painting when facing an empty canvas or a pile of uniformly colored primitives. In this case the user can get inspiration from the colors of another image, such as a photo or a scanned painting. To do this one can browse for an image file, load it into the program, position and scale it interactively and then have the program copy it into the color-buffer. The primitives take up their new color which mostly yields nice variations of color from one primitive to another (see Figure 23).



(a) Original photo. ©Tobias Isenberg. Used with permission.

(b) Points colored according to the photo.

Figure 23. Loading an image into the color-buffer. The primitives, in this case pointillism dots, receive the color of a reference image.

One of the secondary functions of the application is to save the current painting in the form of an image file. This can be done with a simple click on the screenshot button. The image can be saved at any point in time so that different stages of the work can be captured. This allows for playing around with the painting and trying out different variants. During the saving of the screenshot the palette is not being rendered and thus is not contained within the saved image file. Writing an image file is realized using the DevIL image library, which is also used for the loading of interface and primitive textures.

*DevIL is a full-featured, cross-plattform, open-source image library.*

After being requested by two artists, which informally tested the application, layer functionality has been integrated. With the ability to create new and switch between layers, primitives can be stacked on top of each other and manipulated separately. Layers simulate the real painting feature of dried paint. One can apply paint in an already painted area, without affecting the old paint. Being able to chose a layer makes it possible to go back into an old layer of paint and make changes. In case of decorative mosaics, several layers of tiles can be laid on top of each other as seen in [Park 2004]. To realize layer switching, the buffers of the layer currently not active have to be stored on hard drive. Because of the high memory demand for the buffers, it would not be desirable to have multiple sets of buffers in the memory at once. That is why whenever the user selects a different layer than the current one, all buffers of the current layer are stored. If the layer is later on picked again, the buffers are loaded back into the memory.

## 4.3 CLASSES

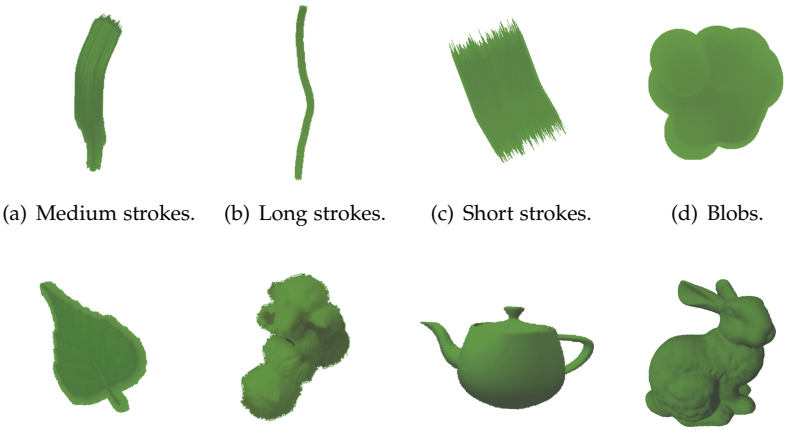
This part highlights details of the implementation and the structure of the program. The main objects that make up the system are the canvas, the primitives, the palette and the tools. They all have to work together and utilize the buffers in certain ways.

### 4.3.1 *Canvas*

The canvas is the environment for all the other components. It holds the buffers and fills them with initial values. The canvas also holds the parameter-buffer. This class is able to save and load entire buffers from the hard drive which is necessary for the layer functionality (see Section 4.2.6). A user can recognize the canvas as a white background taking up the whole screen.

### 4.3.2 *Primitives*

Primitives reside in the canvas. The primitive class reads the buffer values underneath its position. All primitives access the size-, orientation-, color-, motion-, and erase-buffer. It also retrieves information from the parameter-buffer. A primitive has to know, for example, what the current active layer is. Derived from that class are specialized classes for brush strokes, mosaic tiles, and pointillism dots. Each derived class holds a different graphical representation. Brush strokes use a quad with a stroke texture attached to it. It can take up a variety of textures to create different effects in the image (see Figure 24).



(a) Medium strokes. (b) Long strokes. (c) Short strokes. (d) Blobs.

(e) Leafs. (f) Popcorn. (g) Teapots. (h) Stanford bunnys.

Figure 24. Different texture sets for the brush stroke primitive. Every stroke set contains a number of textures similar to the representative examples (colored in green) given here.

Mosaic tiles are beveled quads which are assembled using polygons. In order to make the bevels visable, illumination computation is being used. The mosaic class has the additional task of dodging other tiles and is, therefore, accessing a position-buffer. This buffer contains the positions of all mosaic primitives currently on the canvas. In each render step, every mosaic tile inspects the area it covers in the position-buffer to ascertain if it is overlapping with another tile. If it does, a vector is calculated, which is used to move the tile away from the interfering tile. This feature can also be turned off if the users want to.

Pointillism dots are represented by ellipses. Instead of using scanned point strokes, as described in Section 2.1.6, simple ellipsoidal shaped polygons are used for these primitives. This increases rendering speed and is still sufficient for the pointillism style, because the usual size of a dot is about 9 – 17 pixels. The point class changes its OpenGL shape from a circle to an ellipse according to the speed that is applied when using the orientation-tool. This specialty requires the use of a speed-buffer which contains scalar values delineating the velocity applied at a position. Figure 25 shows an example where elliptic shapes are used

to convey the shape of an object.

Figure 25. Pointillism dots with different shapes. This is an enlarged detail of pointillism dots taking up elliptic shapes to convey shape in the image.

### 4.3.3 *Palette*

The palette class contains all the functionality and elements for the user interface. Its visual representation is shaped like a traditional wooden palette containing buttons, color-swatches and other widgets. If the palette is being touched it highlights its border and the translate only area (see Figure 26 (a)). If a color-swatch or gradient is touched, the palette shows the mixing area to indicate where mixing is possible (see Figure 26 (b)). If a button on the palette is touched it has to pop up the respective widget, e. g. pie menus, and receive the user input. User inputs, namely mode changes or parameters for the tools, are

(a) Translate only area. (b) Mixing area.

Figure 26. Highlighting of certain areas on the palette.

written into the parameter-buffer. The palette also handles sub-objects like color-swatches, color gradients, and the color indicator.

### 4.3.4 *Tool*

The tool class implements all the functionality used to manipulate and paint. With the tool the user can alter size, orientation, color, motion, speed, and erase values in the property buffers. There is only one class for all tool types, which simplifies the program structure.

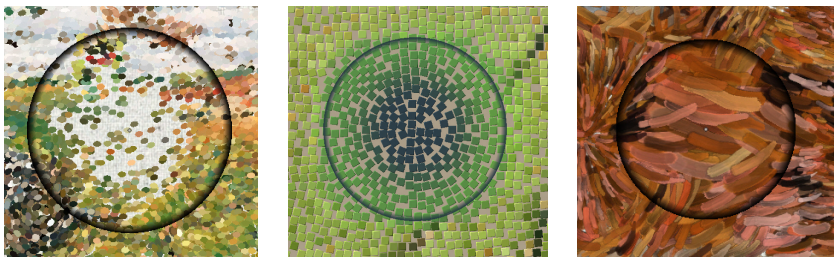
The tool is represented by a circle on the canvas. The circle shape was chosen because it is rotation independent and it applies value changes in the same way no matter where the user stands with respect to the table. The tool diameter can be increased or decreased. This allows for manipulation of a large amount of primitives at once or just a small area. Although the concept of this application is to simultaneously affect several primitives at a time, the user can also work on details at a very low level.

If the canvas surface is touched the tool appears and it starts to write into the necessary buffer. In order to know what values to change in the buffers, it retrieves the current tool type (e. g., size, orientation, color, etc.) from the parameter buffer. It also needs to know what type of primitive it has to create, when the create-tool is active.

## 4.4 OPTIMIZATION

A number of optimization strategies have been implemented to ensure timely animation and a responsive interface.

PRE-COMPUTED ACTIVE-BUFFERS: When working with the tool, one will notice that the new values, for example a new color, are not applied in the same way for every point in the circle. To avoid circle shaped patterns and a sudden step at the tool circle border, new values are applied attenuated (see Figure 27). This means the new value is completely applied at the center of the tool circle and gradually mixed when getting towards the border of the circle. For the color-tool this results in a mixture of the color at the outer area of the circle. To speed up this process of modification of the new values, these values are



(a) Erase-tool. (b) Color-tool. (c) Size-tool.

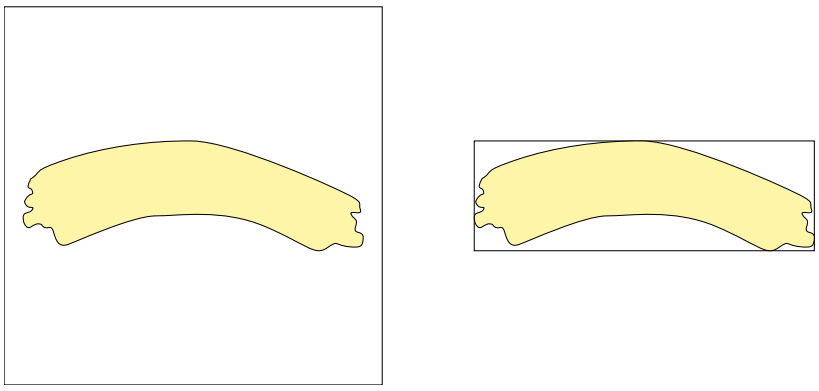
Figure 27. Some examples of gradually applied values.

precomputed and stored in an *attenuation-buffer*. The attenuation-buffer contains values between 0 and 1 and needs to be updated if the tool diameter has changes. Listing 1 contains pseudocode which shows the modification of the values, that are being written into the buffer. This pre-computation is used by almost every tool and ensures a fast mixing of old and new values.

Listing 1. Pseudocode for gradually applied values.

```
for (all positions (x, y) in the tool circle) {
        // get the value from the attenuation buffer at that
            position
        attenuationBufferValue = attenuationBuffer(x, y);
        // get the value from the property buffer at that position
        propertyValue = propertyBuffer(x, y);
        // multiply the new value with the active buffer value
        propertyValue += changeToApply * attenuationBufferValue;
        // write the modified new value back into the property
            buffer
        propertyBuffer(x, y) = propertyValue;
}
```

OPTIMIZED STROKE QUAD: For the brush stroke a simple quad with a texture attached to it is used. The stroke textures are pre-drawn, but they are still sufficient for giving the impression of working with actual strokes. This representation of the strokes has been chosen, because highly detailed simulations at high resolutions are currently hard to realize at interactive rates. The texture is partially transparent so that OpenGL has to perform blending while rendering. Blending combines the color already in the framebuffer with the color of the incoming primitive. That means, objects behind a stroke can be seen where the stroke texture is transparent. Blending, and also the regular rendering of a textured quad, can be an expensive operation, when performed on a vast amount of primitives. That is why the quad size is being reduced to a minimum, in order to decrease rendering time (see Figure 28).



(a) Regular texture quad. (b) Optimized stroke quad.

Figure 28. Optimization of the stroke quad.

To ascertain the necessary quad dimensions the maximum opaque extensions of the actual stroke within the texture are determined. This ensures that only the necessary part of the texture is rendered. The result of the optimization can be seen in Table 1.

| optimization | frames per second |
|---|---|
| before optimization | 14 |
| after optimization | 20 |

Table 1. Result of optimizing the stroke quad size. Framerates are measured in fps with 2000 brush strokes on a 1,680 × 1,050 pixel canvas at a quad size of 64 pixel. For this test the stroke textures from the set in Figure 24 (a) have been used.

## 4.5 A NOVEL PAINTING SYSTEM

The implemented user interface, represented as a palette, is a straightforward collection of widgets that can be moved around and even shared with others. It allows for click-less selection of tools, using widgets such as pie menus and simple sliders, without the need for parameters and text labels. The number of buttons and other elements on the palette has been reduced to a minimum to keep it as clearly arranged as possible. The painting interface also affords the capability to identify more than one input, when using it on large interactive displays, such as a tabletop display. Having this capability, bimanual painting can improve the usability.

There have been other attempts to create familiar interfaces by recreating the appearance of their physical counterparts. One example are software CD-players that look like actual CD-racks and that have the same tiny buttons and low resolution displays common to those devices. It is not clear whether or not these interfaces are feasible for the users on a desktop PC with a WIMP environment. The interface implemented for this system, however, is aimed at large, interactive wall and tabletop displays, where such replications are more feasible. Widgets, such as a wooden palette, can be displayed at its actual size which allows the user to touch it and move it around in a comfortable way.

With the various tools integrated into the system, the user can change attributes of the rendering primitives, such as their size, orientation, and color. Other tools let the user write motion, speed, or erase values into the property buffers to influence their position, shape, or existence. This chapter covered the implementation of the system components and how they work together. In addition, two simple optimizations have been presented that have been carried out to increase the responsiveness of the system.

# 5 DISCUSSION

This last chapter summarizes this work and takes a look at possible extensions and improvements. Example images created with the presented system demonstrate the potential of the paradigm of modeling with rendering primitives applied to SBR. These images also show how users can easily create expressive digital paintings in a short amount of time. Because this system was mainly designed to also support large displays by incorporating strategies such as click-less interfaces this last chapter compares its usage on such displays and standard desktop monitors.

## 5.1 APPLICATION ON DIFFERENT DISPLAYS

In this part the results of the practical use on different display devices are presented. The photos in Figure 29 and Figure 30 show a user painting on a horizontal tabletop display and on a vertical wall display. These large, high-resolution displays are assembled using a DViT BOARD™ by SMART Technologies and a rear projection. Both display devices offer, compared to a desktop PC, two simultaneous inputs at the same time using DViT technology. This allows the user to use both hands for the interaction with the system.

To ascertain that the presented system is in fact interactive, framerates to reflect the rendering speed were measured. A standard desktop

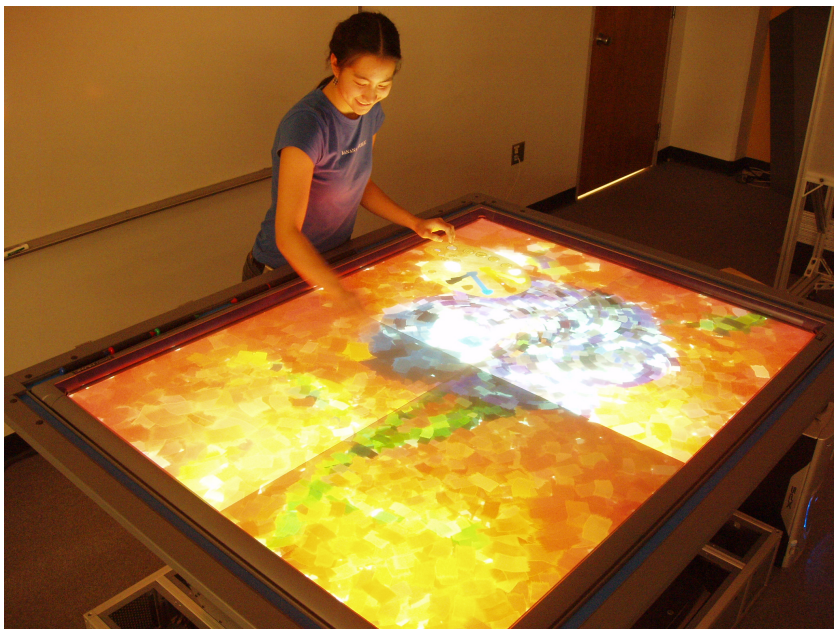


Figure 29. Painting on a horizontal tabletop display.

Figure 30. Painting on a vertical wall display.

monitor (1,400 × 1,050 pixels; 1.47 million pixels) and a large, high-resolution display (2,800 × 2,100 pixels; 5.88 million pixels), with a physical size of (146 cm × 110 cm), have been used to compare two different devices. To make the setups comparable, both displays were driven by the same machine:

- 3 GHz dual core processor
- 2 GB RAM
- two 512 MB nVIDIA GeForce 7900 GTX (using a soft SLI connection)

Brush strokes and pointillism ellipses where 64 pixels long and mosaic tiles had a size of 16 × 16 pixels. The results of the measurement in frames per second (fps) can be seen in Table 2 for the desktop display and in Table 3 for the tabletop display.

| primitive count | brush strokes | mosaic tiles | points |
|---|---|---|---|
| 1000 | 84 | 24 | 159 |
| 2000 | 63 | 13 | 86 |
| 4000 | 43 | 7 | 32 |
| 8000 | 26 | - | 16 |
| 16000 | 15 | - | 8 |

Table 2. Framerates in fps on a desktop monitor. Display resolution: 1,400 × 1,050 pixels.

| primitive count | brush strokes | mosaic tiles | points |
|---|---|---|---|
| 1000 | 29 | 17 | 31 |
| 2000 | 24 | 12 | 27 |
| 4000 | 19 | 7 | 21 |
| 8000 | 13 | 4 | 15 |
| 16000 | 8 | 2 | 10 |

Table 3. Framerates in fps on a tabletop display. Display resolution: 2,800 × 2,100 pixels.

For the desktop setting, 1000 brush strokes were enough to fill the entire canvas. On the tabletop display 4000 strokes were sufficient. There are no results for more than 4000 mosaic tiles on the desktop setting, because this amount of tiles already fills the canvas and additional tiles are just being pushed from the canvas surface. Note that there is an additional bottleneck besides the rendering on the interactive surfaces. The touch input from the DViT Board is less responsive than the input from a mouse on a desktop PC.

It is obvious that the computation time for the mosaic tiles is too high for a larger amount of tiles on the canvas. The dodging algorithm could be improved to regain responsive interaction rates using graphics hardware as in [Hoff III et al. 1999]. Nevertheless, the results are evidence for the system to be interactive even at with a large number of brush strokes and point primitives on the canvas.

## 5.2 EXAMPLE IMAGES

The series of images in Figure 31 shows an example painting creation using the delineated system. We start off by creating a rough background wash using very large brush strokes. In this example we would like to use the colors from a glowing sunset to color our strokes. This can easily be done by loading the image shown in Figure 31 (a) as described in Section 4.2.6. This gives us an idea of the arrangement of the painting (see Figure 31 (b)). Now we create a second layer, so that we can leave the background as it is and work with smaller strokes on top of it. By this, we also do not have to fill the entire screen on each layer. Let us assume we want to work out some detail for the sky and the smoke part. We simply create strokes in these parts, shrink their size until satisfied and then fix their orientation. We would like to give the smoke a wild look, which can easily be done with one of the orientation modes available. We also want to transform the original uniform sky into a furious one. To color our strokes we once again load our photo into the program (Figure 31 (c)). We have decided that some parts require special emphasis and we therefore add a third layer (see Figure 31 (d)). In this last step we apply very fine strokes for the sun and orient them in a circle wise manner.

(a) Original photo used to color the strokes with. © Tobias Isenberg. Used with permission.

(b) A background wash is created using very large strokes and applying the photo for coloring the strokes.

(c) In a second layer smaller brush strokes are added to induce wild smoke and a stormy sky.

(d) A third layer is added to work out detail in areas where emphasis is desired.

Figure 31. Image creation step by step.

Figure 32 gives another example of painting with the brush strokes. Longer brush strokes have been used to visualize the water and its reflections. The shore was created using pointillism dashes which are especially suitable for very detailed areas such as trees and grass. The long strokes did not work for the upper part so they have been deleted and replaced. The flexibility of the create-erase-tool allows the user to experiment with different types of strokes for a particular part of the image.

Figure 32. Turquoise lake. Original photo © Petra Neumann. Used with permission.

The painting in Figure 33 was done utilizing very short and more thicker strokes. They work very well for the bricks of the pyramid and they add a distinct playful look to the sky. For this painting a picture showing pyramids was loaded into the color-buffer but the sky was repainted. With the color-tool and the possibility to increase or decrease the tool diameter, the user can recolor large parts of the image or apply a new color only on certain spots.

In Figure 34 leaves have been used as strokes. In this case no color image was used to color the strokes. The color was applied manually. The motif was inspired by works by ANDY GOLDSWORTHY ROWAN. The image was painted by Petra Neumann. The photo in Figure 35 shows a user on a wall display painting the leaf-image in Figure 34.

The tiger lily in Figure 36 is a very nice example for the application of pointillism dots. This image was created using an actual photo of a tiger lily. Large green blotches are used for the background and very small ones for the details of the flower. The car in Figure 37 was painted using medium brush strokes and an orientation that tries to make the car look fast. The background is only roughly adumbrated which lets the car stand out in front. Figure 38 shows an example for a decorative mosaic pattern. Notice how the tiles in the center

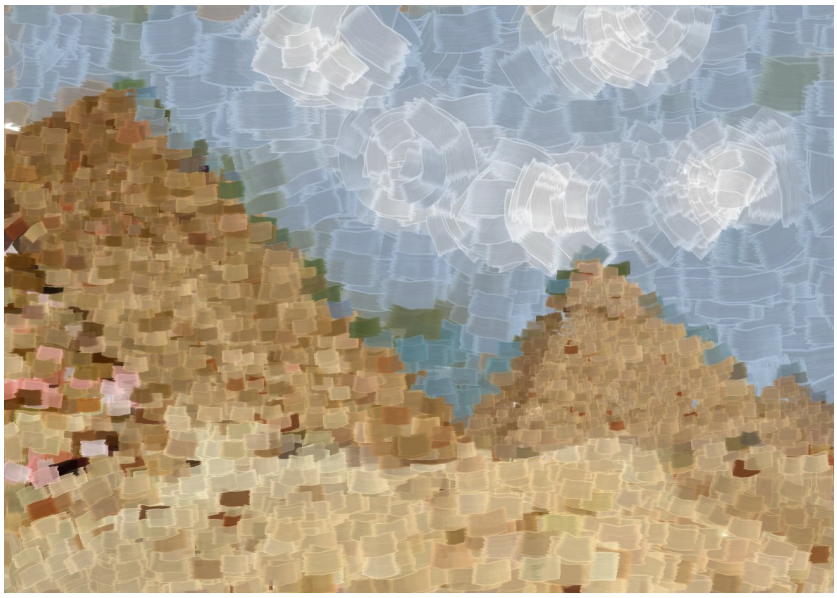


Figure 33. Pyramids. Original photo © Tobias Isenberg. Used with permission.

are aligned in a circular manner. This was accomplished by simply applying the orientation-tool with an orbit orientation type. Such a manipulation would be rather hard to do when using a traditional NPR filter. Figure 39 is an interesting example for the use of popcorn strokes. This image was also created by Petra Neumann. The plant in Figure 40 shows the result of using multiple reference images to color the strokes. This image consists of rather unusual blob strokes, but they yield an interesting effect.

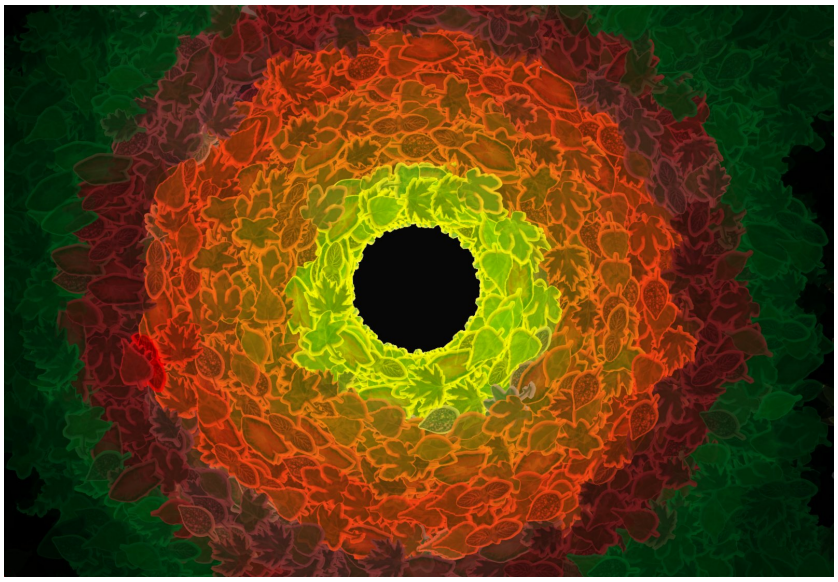


Figure 34. Free-hand painting with leaf primitives © Petra Neumann. Used with permission.

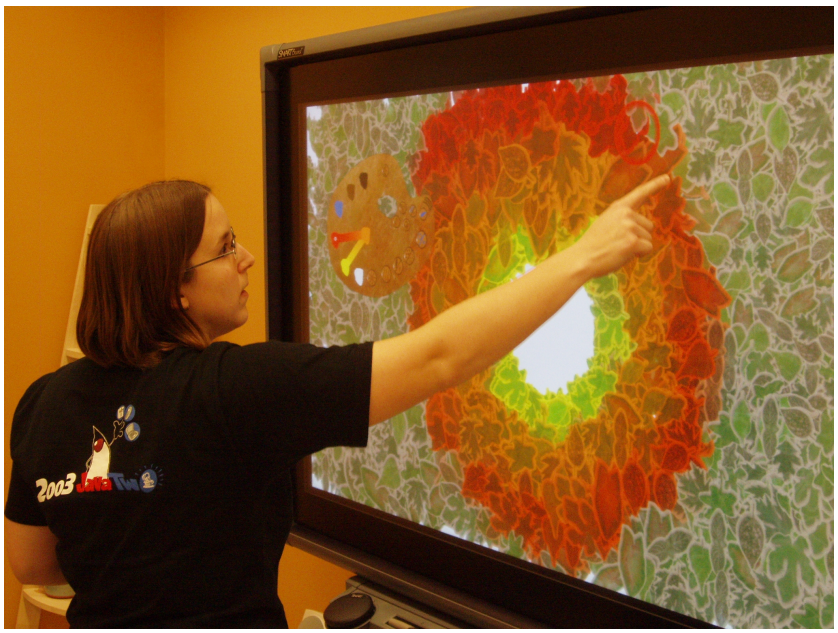


Figure 35. Free-hand painting at a wall display.

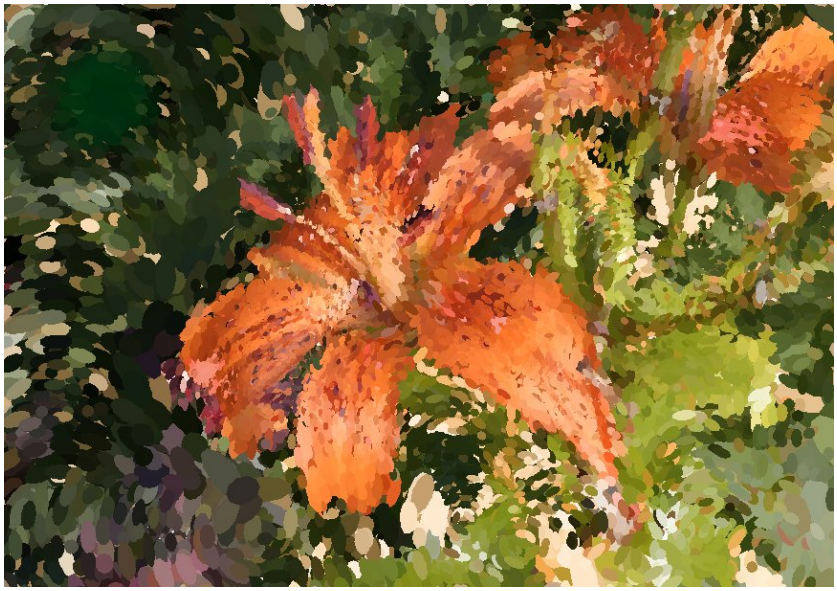


Figure 36. Tiger lily. Original photo © Edward Tse. Used with permission.

Figure 37. Porsche 386.

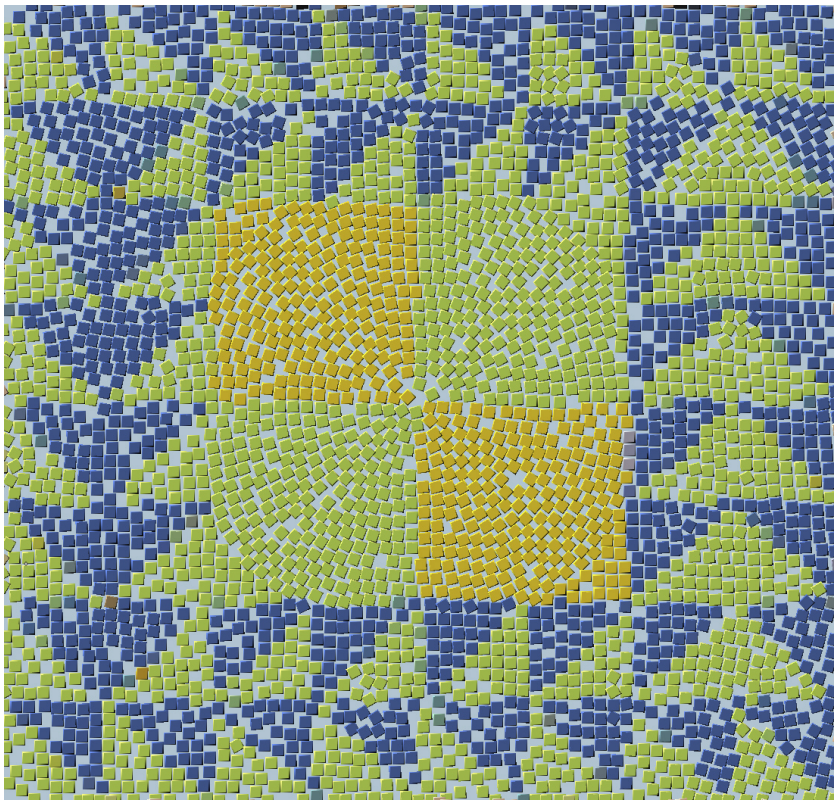


Figure 38. Decorative mosaic pattern.

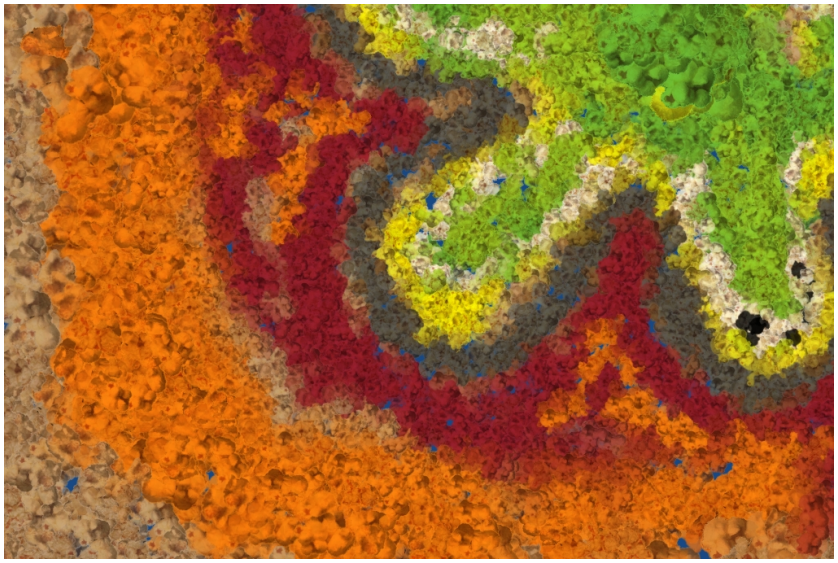


Figure 39. Pop(corn)art. © Petra Neumann. Used with permission.

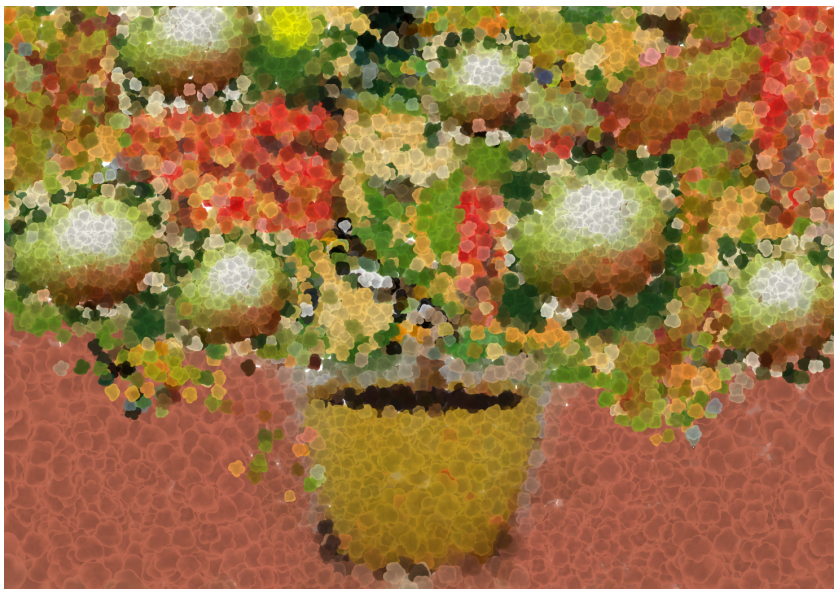


Figure 40. Apple “plant”.

Figure 41. Cherries.

Figure 42. Foxglove by © Sheelagh Carpendale. Used with permission.

Figure 43. Barn with cornfield.

## 5.3 CONCLUSION

The presented system implements the concept of modeling with rendering primitives. This opens up the black box of rendering and makes it possible to influence the primitives and to modify aspects as desired. The users are able to create non-photorealistic imagery without being constrained by a non-transparent algorithm. The system provides an easy to use interface that is especially well suited for the application on large, interactive, direct-touch displays. The user interacts with the primitives through a buffer stack that can be manipulated with various tools. The buffer concept that has been utilized allows for responsive interaction even with a large number of primitives. Artistic styles such as oil painting, pointillism, and decorative mosaics can be imitated, even though the focus lies on interaction and not on a realistic simulation. This work is not only directed towards experienced artists, who want to experiment with a novel way of creating and interacting with images, but also novices who want to take on painting as entertainment. Because of the simplified interface, one could even think about using it as a painting application for kids.

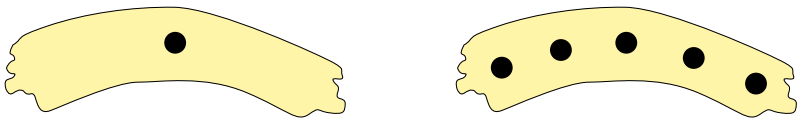
## 5.4 FUTURE WORK

The presented project can be extended in various ways and might open up further investigation into certain particulars.

PRIMITIVES: Three artistic styles common to non-photorealistic rendering have been integrated into this system. Another possible style suitable for this approach would be stippling. This technique also requires input from the user and would benefit from the concurrency of modeling and rendering.

One could also offer a larger variety of primitive types. The mosaic tiles could take up the shape of a triangle, a diamond or any other arbitrary shape. For the brush strokes an extension could be developed that would allow the users to paint their own set of strokes which they can later on use in the program. The representation of brush strokes could be changed to some form of spline where each individual node reads buffer values at its position (see Figure 44). Such a stroke could be constructed out of several quads, whose size could then be determined by the size-buffer value at each node position. Having different color or orientation values at each node would be even more interesting. One could also incorporate dynamic primitives such as graftals [Kowalski et al. 1999; Markosian et al. 2000], which can algorithmically generate geometry in the rendering process.

The visual appearance of the primitives could be improved in several ways. One could use more than one texture to apply, for example, bump maps for the brush strokes or reflection maps for the mosaic tiles. This may also be combined with the utilization of a shader language to speed up rendering and keep the painting at interactive frame rates. To make the mosaic tiles look more natural, they could overlap each other and then being clipped where they overlap. Although this is usually done in real mosaics, this would require a higher computational effort



(a) Current stroke with one buffer-access position.

(b) Possible stroke with multiple buffer-access positions.

Figure 44. A possible extension for the brush strokes. A long stroke could access the buffer at multiple positions.

and ought to be considered as a post-processing step.

As already mentioned in Section 5.1 one could implement a more sophisticated algorithm for the placement of mosaic tiles. The current solution works with a moderate amount of tiles but is not very responsive with a high number of mosaic primitives. Nevertheless it demonstrates the idea of interactive mosaics and is enough to create a variety of images. Admittedly, this might introduce a certain degree of automation, which would be a step away from the idea of full user control.

FUNCTIONALITY: The color mixing described in Section 4.2.5 could be further extended by allowing the connection of two existing color-swatches or creating a new blot on a color gradient and therewith provoke a gradient between a color-swatch and a color gradient.

Another thing that has been requested by several test-users is the ability to save and load image stages. Although creating a painting can be done quickly with this system, some users tend to work on a very low level of the image an manipulate only a few strokes at a time. This may result in a longer painting times where the user wants to be able to save their current stage, close the application and continue to work at a later date. Because the canvas is already able to save and load buffers, this would not be difficult to realize. In addition, the primitives with their position and internal properties would have to be stored.

To offer high quality output that is independent from the display resolution one could consider to use vector graphics. There are tools available that convert OpenGL primitives into vector graphics [Tobias Isenberg and Sousa 2005] (e. g., GPL, GL2PS, or ClibPDF), which could be used for decorative mosaics and pointillism images.

INTERACTION: The system has been deployed on an interactive display affording two simultaneous touches. This would allow two people to use the application at once sharing the palette and the tool. Using a device with even more inputs would facilitate collaborative painting where several users work on different parts of the painting. The system is already capable of creating and managing multiple palettes and multiple tools. This could be an interesting painting experience and would surely be interesting to investigate in.

The buffers could also be used for purposes other than intended.

It would be possible for a primitive to combine two or more buffer value to create some interesting effects. The color could, for example, be influenced by the orientation of a primitive or the size could be the result of a combination of the color and the speed at a certain buffer position. Images could be loaded into buffers other than the color-buffer and there could be tools that do image processing on the buffers.

Apitz, Georg, and Guimbretiere, Francois. 2004. CrossY: a crossing-based drawing application. In *Proceedings of the ACM Symposium on User Interface Software and Technology*, Pens & sketching, 3–12. (Cited on pages 12 and 14.)

Battiato, S., di Blasi, G., Farinella, G. M., and Gallo, G. 2006. A survey of digital mosaic techniques. In *Eurographics Italian Chapter Conference*, Eurographics, G. Gallo, S. Battiato, and F. Stanco, Eds., 129–135. (Cited on page 10.)

Baxter III, W. V., Scheib, V., Lin, M. C., and Manocha, D. 2001. DAB: interactive haptic painting with 3D virtual brushes. In *SIGGRAPH*, 461–468. (Cited on page 7.)

Bier, E. A., Stone, M. C., Pier, K., Buxton, W., and DeRose, T. D. 1993. Toolglass and Magic Lenses: The See-Through Interface. In *SIGGRAPH93*, ACM Press, New York, 73–80. (Cited on page 14.)

Buchin, K., and Walther, M. 2003. Hatching, Stroke Styles & Pointillism. In *ShaderX² – Shader Tips and Tricks*, W. Engel, Ed. Wordware Publishing, Inc., 340–347. (Cited on page 11.)

Burns, M., Klawe, J., Rusinkiewicz, S., Finkelstein, A., and DeCarlo, D. 2005. Line Drawings from Volume Data. 512–518. (Cited on page 17.)

Byrne. 1993. Using icons to find documents: Simplicity is critical. In *Proceedings of ACM INTERCHI'93 Conference on Human Factors in Computing Systems*, Searching: Tools and Strategies, 446–453. (Cited on page 16.)

Cohen, M., Puech, C., Sillion, F., Haeberli, P., and Segal, M., 1993. Texture mapping as a fundamental drawing primitive, Sept. 09. (Cited on page 8.)

Curtis, C. J., Anderson, S. E., Seims, J. E., Fleischer, K. W., and Salesin, D. H. 1997. Computer-Generated Watercolor. ACM Press/ACM SIGGRAPH, New York, T. Whitted, Ed., 421–430. (Cited on page 17.)

DeCarlo, D., and Santella, A. 2002. Stylization and abstraction of photographs. *ACM Trans. Graph 21*, 3, 769–776. (Cited on page 7.)

Deussen, O., Hiller, S., van Overveld, C. W. A. M., and Strothotte, T. 2000. Floating Points: A Method for Computing Stipple Drawings. 40–51. (Cited on pages 7 and 17.)

Ebert, D., and Rheingans, P. 2000. Volume Illustration: Non-Photorealistic Rendering of Volume Models. In *Proceedings Visualization 2000*, T. Ertl, B. Hamann, and A. Varshney, Eds., IEEE Computer Society Technical Committee on Computer Graphics, 195–202. (Cited on page 17.)

Elber, G., and Wolberg, G. 2003. Rendering Traditional Mosaics. *The Visual Computer 19*, 1 (mar), 67–78. (Cited on page 9.)

Epps, J., Lichman, S., and Wu, M. 2006. A Study of Hand Shape Use in Tabletop Gesture Interaction. In *Proc. of ACM CHI 2006*, ACM Press, New York, 748–753. (Cited on page 14.)

Fischer, J., and Bartz, D. 2005. A Pointillism Style for the Non-Photorealistic Display of Augmented Reality Scenes. Tech. Rep. WSI-2005-05, Wilhelm Schickard Institute for Computer Science, Graphical-Interactive Systems (WSI/GRIS), University of Tübingen, Germany, May. (Cited on page 11.)

Foster, K., Jepp, P., Wyvill, B., Sousa, M. C., Galbraith, C., and Jorge, J. A. 2005. Pen-and-Ink for BlobTree Implicit Models. *EUROGRAPHICS2005 24*, 3 (Sept.), 267–276. (Cited on page 17.)

Gossett, N., and Chen, B. 2004. Paint Inspired Color Mixing and Compositing for Visualization. In *Proc. of IEEE InfoVis 2004*, 113–117. (Cited on page 31.)

Guiard, Y. 1987. Assymetric division of labor in human skilled bimanual action: The kinematic chain as model. *Journal of Motor Behaviour 19*, 4, 486–517. (Cited on page 13.)

Guimbretière, F., and Winograd, T. 2000. FlowMenu: combining command, text, and data entry. In *Proceedings of the 13th Annual Symposium on User Interface Software and Technology (UIST-00)*, 213–216. (Cited on page 12.)

Haeberli, P. E. 1990. Paint by numbers: Abstract image representations. In *Computer Graphics (SIGGRAPH '90 Proceedings)*, F. Baskett, Ed., vol. 24, 207–214. (Cited on page 5.)

Hausner, A. 2001. Simulating Decorative Mosaics. E. Fiume, Ed., ACM SIGGRAPH, 573–580. (Cited on page 9.)

Hertzmann, A. 1998. Painterly Rendering with Curved Brush Strokes of Multiple Sizes. ACM Press/ACM SIGGRAPH, New York, M. Cohen, Ed., 453–460. (Cited on pages 8, 17, and 27.)

Hertzmann, A. 2001. Paint By Relaxation. In *Proc. of Computer Graphics International 2001*, IEEE Computer Society Press, Los Alamitos, 47–54. (Cited on page 9.)

Hertzmann, A. 2003. A Survey of Stroke-Based Rendering. *IEEE Computer Graphics and Applications 23*, 4 (July/Aug.), 70–81. (Cited on page 7.)

Hoff III, K. E., Keyser, J., Lin, M., Manocha, D., and Culver, T. 1999. Fast computation of generalized Voronoi diagrams using graphics hardware. *Computer Graphics 33*, Annual Conference Series, 277–286. (Cited on page 41.)

Hopkins, D. 1991. The design and implementation of pie menus. *Dr. Dobb's Journal of Software Tools 16*, 12 (dec), 16–26, 94. (Cited on page 14.)

HORTON, AND WILLIAM. 1994. *The Icon Book: Visual Symbols for Computer Systems and Documentation*. John Wiley & Sons. OCLC 28962614. (Cited on page 16.)

ISENBERG, T., MIEDE, A., AND CARPENDALE, S. 2006. A Buffer Framework for Supporting Responsive Interaction in Information Visualization Interfaces. In *Proc. of C5 2006*, IEEE Computer Society, Los Alamitos, 262–269. (Cited on page 19.)

KALNINS, R. D., MARKOSIA, 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. In *SIGGRAPH 2002 Conference Proceedings*, ACM Press/ACM SIGGRAPH, J. Hughes, Ed., Annual Conference Series, 755–762. (Cited on pages 7 and 17.)

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. ACM Press, New York, 433–438. (Cited on page 50.)

KRUGER, R., CARPENDALE, S., SCOTT, D., S., AND TANG, A. 2005. Fluid integration of rotation and translation. In *Proceedings of ACM CHI 2005 Conference on Human Factors in Computing Systems*, vol. 1 of *Physical interaction*, 601–610. (Cited on page 23.)

LANG, D., FINDLATER, L., AND SHAVER, M. 2003. CoolPaint: Direct Interaction Painting. In *Poster Proceedings of User Interface Software and Technology (UIST) 2003*. (Cited on page 5.)

LANK, E., RUIZ, J., AND COWAN, W. 2006. Concurrent bimanual stylus interaction: a study of non-preferred hand mode manipulation. In *GI '06: Proceedings of the 2006 conference on Graphics interface*, Canadian Information Processing Society, Toronto, Ont., Canada, Canada, 17–24. (Cited on pages 14 and 17.)

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. ACM Press, New York, 59–64. (Cited on page 50.)

MASON, K., AND CARPENDALE, S. 2001. Artist-Driven Expressive Graphics. Blackwell Publishers, Oxford, UK, Eurographics Association, 87–96. (Cited on page 7.)

MASON, K., AND CARPENDALE, S. 2001. Expanding the Expressive Palette. Tech. Rep. 2001-685-08, Department of Computer Science, University of Calgary, Canada. (Cited on page 7.)

MASON, K., DENZINGER, J., AND CARPENDALE, M. S. T. 2004. Negotiating gestalt: Artistic expression and coalition formation in multiagent systems. In *AAMAS*, IEEE Computer Society, 1350–1351. (Cited on page 7.)

MASON, K. M. 2006. *A Framework for Element-Based Computer Graphics*. PhD thesis, University of Calgary, Canada. (Cited on pages 7, 17, 18, and 27.)

Meier, B. J., Spalter, A. M., and Karelitz, D. B. 2004. Interactive color palette tools. *IEEE Computer Graphics and Applications 24*, 3, 64–72. (Cited on page 31.)

Meier, B. J. 1996. Painterly rendering for animation. In *SIGGRAPH 96 Conference Proceedings*, Addison Wesley, H. Rushmeier, Ed., Annual Conference Series, ACM SIGGRAPH, 477–484. held in New Orleans, Louisiana, 04-09 August 1996. (Cited on page 8.)

Meier, B. 1999. Computers for artists who work alone. *SIGGRAPH Comput. Graph. 33*, 1, 50–51. (Cited on pages 7 and 15.)

Miede, A. 2006. *Realizing Responsive Interaction for Tabletop Interaction Metaphors*. Master's thesis, Otto-von-Guericke-Universität Magdeburg. (Cited on page 19.)

Morgan, W., Lee, F. J., and Kratz, L. A. 2006. Explorations in Gameplay Using Bimanual Mice. (Cited on page 14.)

Moscovich, T., and Hughes, J. F. 2006. Multi-Finger Cursor Techniques. A K Peters, Ltd., Wellesley, MA, USA, 1–7. (Cited on page 14.)

Mullet, K., and Sano, D. 1995. *Designing Visual Interfaces*. Prentice Hall. (Cited on page 15.)

Newman, W. M., and Lamming, M. G. 1995. *Interactive System Design*. Addison-Wesley, Reading, Massaachusetts. (Cited on page 13.)

Ni, T., Schmidt, G. S., Staadt, O. G., Livingston, M. A., Ball, R., and May, R. 2006. A survey of large high-resolution display technologies, techniques, and applications. In *Proceedings of IEEE Virtual Reality 2006*, IEEE Computer Society. (Cited on page 2.)

Olsen Jr., D. R. 1998. *Developing User Interfaces*. Morgan Kaufmann. (Cited on pages 15 and 16.)

Park, Y. S., and Yoon, K. H. 2004. Adaptive Brush Stroke Generation for Painterly Rendering. 65–68. (Cited on pages 1, 17, and 26.)

Park, Y., and Yoon, K.-H. 2006. Motion Map Generation for Maintaining the Temporal Coherence of Brush Strokes. In *Proceedings of the First International Conference on Computer Graphics Theory and Applications (GRAPP 2006, February 25–28, 2006, Setúbal, Portugal)*, INSTICC – Institute for Systems and Technologies of Information, Control and Communication, Setúbal, Portugal, J. Braz, J. A. Jorge, M. Dias, and A. Marcos, Eds., 158–167. (Cited on page 26.)

Park, J. W. 2004. Mosaic for stackable objects. In *SIGGRAPH '04: ACM SIGGRAPH 2004 Sketches*, ACM Press, New York, NY, USA, 1. (Cited on page 33.)

Preece, J. 1994. *Human-Computer Interaction*. Addison-Wesley, Reading, MA, USA. (Cited on page 13.)

Salisbury, M. P., Wong, M. T., Hughes, J. F., and Salesin, D. H. 1997. Orientable textures for image-based pen-and-ink illustration. In *SIGGRAPH 97 Conference Proceedings*, Addison Wesley, T. Whitted,

Ed., Annual Conference Series, ACM SIGGRAPH, 401–406. (Cited on pages 6 and 17.)

SANTELLA, A., AND DECARLO, D. 2002. Abstracted Painterly Renderings Using Eye-Tracking Data. ACM Press, New York, A. Finkelstein, Ed., 75–82. (Cited on page 7.)

SCHLECHTWEG, S., GERMER, T., AND STROTHOTTE, T. 2005. RenderBots—Multi Agent Systems for Direct Image Generation. 137–148. (Cited on pages 8 and 17.)

SCHWARZ, M., ISENBERG, T., MASON, K., AND CARPENDALE, S. 2007. Modeling with Rendering Primitives: An Interactive Non-Photorealistic Canvas. Tech. Rep. 2007-851-03, Department of Computer Science, University of Calgary, Canada, Feb.

SCHWARZ, M., ISENBERG, T., MASON, K., AND CARPENDALE, S. 2007. Modeling with Rendering Primitives: An Interactive Non-Photorealistic Canvas. *Proceedings of the Fifth International Symposium on Non-Photorealistic Animation and Rendering (NPAR 2007, August 4–5, 2007, San Diego, California, USA) 26*, 3 (Aug.). Submitted.

SEMET, Y., AND DUR, F., 2004. An Interactive Artificial Ant Approach to Non-Photorealistic Rendering, Apr. 27. (Cited on page 18.)

SMITH, A. R. 1978. Paint. Technical Memo 7, NYIT Computer Graphics Lab, July. (Cited on page 5.)

SMITH, A. R., 1979. Table paint. Published as Tutorial Notes at SIGGRAPH 80 and SIGGRAPH 81. (Cited on page 5.)

SMITH, A. R. 2001. Digital Paint Systems: An Anecdotal and Historical Overview. *IEEE Annals of the History of Computing 23*, 2, 4–30. (Cited on page 6.)

STACEY D. SCOTT, S. C., AND HABELSKI, S. 2005. Storage bins: Mobile storage for collaborative tabletop displays. *Computer Graphics and Applications 25*, 4, 58–65. (Cited on page 23.)

STROTHOTTE, T., AND SCHLECHTWEG, S. 2002. *Non-Photorealistic Computer Graphics. Modeling, Animation, and Rendering*. Morgan Kaufmann Publishers, San Francisco. (Cited on page 1.)

TAN, D. S., GERGLE, D., SCUPELLI, P. G., AND PAUSCH, R. 2003. With similar visual angles, larger displays improve performance on spatial tasks. In *CHI 2003 Conference on Human Factors in Computing Systems*. (Cited on page 12.)

TOBIAS ISENBERG, SIMON NIX, M. S. A. M., AND CARPENDALE, S. 2007. Mobile spatial tools for fluid interaction. In *In Proceedings of the 20th Annual ACM Symposium on User Interface Software and Technology (UIST 2007, October 7–10, 2007, Newport, Rhode Island, USA)*, ACM Press.

Tobias Isenberg, M. S. T. C., and Sousa, M. C. 2005. Breaking the pixel barrier. In *Proceedings of the First Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging 2005 (May 18-20, 2005, Girona, Spain)*, Eurographics Association, 41–48. (Cited on page 51.)

Tyndiuk, F., Thomas, G., Lespinet-Najib, V., and Schlick, C. 2005. Cognitive comparison of 3D interaction in front of large vs. small displays. In *Proceedings of the ACM Symposium on Virtual Reality Software and Technology, VRST 2005, Monterey, CA, USA, November 7-9, 2005*, ACM, G. Singh, R. W. H. Lau, Y. Chrysanthou, and R. P. Darken, Eds., 117–123. (Cited on page 12.)

Vanderhaeghe, D., Barla, P., Thollot, J., and Sillion, F. X. 2006. A Dynamic Drawing Algorithm for Interactive Painterly Rendering. In *ACM SIGGRAPH 2006 Conference Abstracts and Applications*, ACM Press, New York. (Cited on page 17.)

Wyvill, B., Foster, K., Jepp, P., Schmidt, R., Sousa, M. C., and Jorge, J. A. 2005. Sketch Based Construction and Rendering of Implicit Models. In *Proceedings of the First Eurographics Workshop on Computational Aesthetics in Graphics, Visualization and Imaging 2005 (May 18–20, 2005, Girona, Spain)*, Eurographics Association, Aire-la-Ville, Switzerland, L. Neumann, M. S. Casasayas, B. Gooch, and W. Purgathofer, Eds., 67–74. (Cited on page 17.)

Xu, H., and Chen, B. 2004. Stylized Rendering of 3D Scanned Real World Environments. ACM Press, New York, A. Hertzmann and C. Kaplan, Eds., 25–34. (Cited on page 17.)

Xu, H., Nguyen, M. X., Yuan, X., and Chen, B. 2004. Interactive Silhouette Rendering for Point-Based Models. In *Proceedings of the 2004 Eurographics Symposium on Point-Based Graphics (SPBG'04, Zurich, Switzerland, June 2–4, 2004*, EUROGRAPHICS Association, 13–18. (Cited on page 17.)

Yang, H.-L., and Yang, C.-K. 2006. A non-photorealistic rendering of seurat's pointillism. In *ISVC (2)*, 760–769. (Cited on pages 10 and 11.)

# DECLARATION

I declare that this thesis is my own unaided work, and hereby certify that unless stated, all work contained within this paper is my own. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

*Magdeburg, April 2007*

Martin Schwarz