USER INTERACTIONS FOR SCALABLE FREEHAND SKETCHING

MENNO NIJBOER

university of groningen

Department of Mathematics and Computing Science
Faculty of Computer Science
University of Groningen

Supervisors:
Dr. Tobias Isenberg
Moritz Gerl

February 2009

ABSTRACT

Technological improvements the last decade have made digital drawing a very effective method for artistic expression. Thanks to more widely available and affordable peripheral devices such as pen tablets, an increasing number of artists are creating digital works. However, many fundamental interactions of modern day drawing systems have seen little change in over twenty years. Therefore, it is time to investigate what kind of innovation can be brought to these interactions with the computational power of today's desktop computers.

This thesis shows the development, implementation and evaluation of novel canvas and stroke interactions. A novel way to orientate the canvas through gestures is presented. This is a modeless, effective way for artists to use the canvas which takes little time to become proficient in. A new interaction tool gives the artist intuitive control over stroke shapes by utilizing the brush stroke paradigm which artists are very familiar with. The tool can be used to freely shape lines with very few restrictions.

Methods to create scalable line art from freehand drawing input are discussed. A technique to improve the visual quality of lines is also presented. A global theme of the thesis is the creation of a visually appealing interface which is minimal, yet effective. Finally, these concepts are evaluated through a case study and an informal user study. Results show that the key concepts presented here can be useful additions to the workflow of an artist.

ACKNOWLEDGMENTS

During the course of this thesis project I had the good fortune to have the support of many people. I would like to thank the following people in particular:

Tobias Isenberg, whose feedback, supervision and contagious cheerfulness helped me complete this project.

Moritz Gerl, who diligently proofread my writing, tested the implementation and stayed enthusiastic during the project.

All my close friends, who kept me sane during the long days of coding and writing. You know who you are!

Hugh Thomas, Paul Mikulecky, Sahal Merchant, Avik Kumar Maitra and all the other people who tried the software and gave me valuable feedback.

André Miede, for his latex template, which was used with this thesis.

And of course my parents, for their endless moral support and for making sure I always have a roof over my head.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

NPR Non-Photorealistic Rendering

CAVE Cave Automatic Virtual Environment

SBR Stroke Based Rendering

BSM Brush Stroke Models

GPU Graphics Processing Unit

UI User Interface

UCD User-Centered Design

LOD Level of Detail

BSP Binary Space Partioning

OS Operating System

API Application Programming Interface

GUI Graphical User Interface

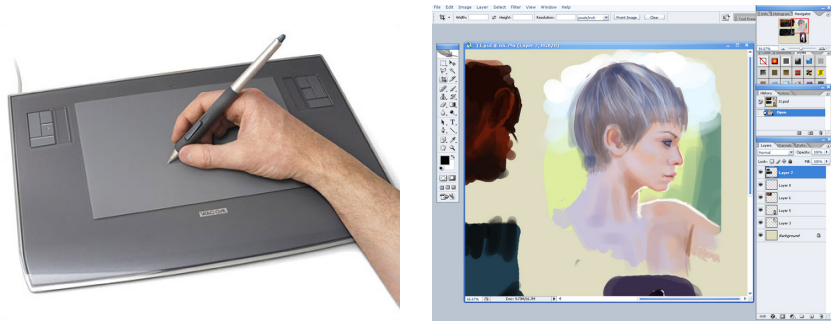
STL Standard Template Library

1 INTRODUCTION

Improvements in computational power and peripheral devices during the last decade have made digital drawing not only a possibility, but a very effective method for artistic expression. Unburdened by the constraints imposed on physical media, such as the high penalty of mistakes and the room needed to store an artists easel, digital painting and drawing are an excellent option for both starting and professional artists alike. Currently, with the wide availability of pen tablets and drawing software, more and more artists are turning to the computer to create their works.

1.1 MOTIVATION

With this increasing popularity of digital drawing it is time to take a fresh look at some of the interactions which have seen very little innovation since the very first drawing programs. This lack of improvement has lead to current programs using methods that were designed for very limited hardware and input devices that are now considered archaic. The question is, therefore, whether we can, with todays powerful computers, improve upon this.



(a) (b)

Figure 1: (a) A Wacom pen tablet [Wacom]. (b) Adobe Photoshop [Adobe, 1990] used to perform digital painting.

In particular, the interfaces of modern drawing applications are very complex. This can lead to a steep initial learning curve. Another problem is the high cost of switching tools, as this typically requires a mode change. These interface issues can be troublesome and disheartening to the artist using the tool. Another point of interest is the lack of control over strokes once they have been deposited on the canvas, or the very unintuitive fashion in which this control can be exercised. Often the only options available in modern systems are the eraser or cumbersome curve shaping controls.

Furthermore, many drawing systems expose a great deal of technical details to the user. For beginners and artists that want to get into digital

art, but have little experience with computers, this can create a high reluctance in learning the ins and outs of a certain drawing application. Such aspects of a drawing system should be transparent to the user.

1.2 CONTRIBUTION

This thesis addresses a small set of important drawing interactions and explores novel ways in which to perform or extend them. The goal is to develop interesting alternatives to the established methods.

To accomplish this goal, several new interactions and interface mechanics were developed. The first of these is a gesture based way to control the placement and orientation of the canvas. This approach is easy to understand, and requires only minimal instruction for a user to become proficient. The interactions also require no mode changes. New possibilities for stroke editing have also been researched. This has led to a brush based edit tool which can redefine the local shape of a stroke. The tool is an intuitive and responsive way in which to manipulate lines. Furthermore, a new gesture based approach to selection transformations was created. This allows all standard transformation operations to be available in one selection frame without the need for mode changes.

A way to create scalable artwork from freehand drawing input was researched. Scalability without loss of quality has several benefits. Artists can reuse small thumbnail sketches as a basis for final works, and are no longer restricted to image resolutions. This alleviates some of the troubles of sharing and printing line art.

(a) (b)

Figure 2: (a) A user working with the implementation. (b) The implementation, used to work on a magnified part of a sketch.

An evaluation conducted towards the end of the project showed that many of the novel interactions were well received. Especially the simple, gesture-based controls were unanimously perceived as useful. Interactions such as the ability to alter the stroke shape were found to be interesting, especially as a method for experimentation. Overall, many of the concepts presented in this thesis are interesting for future research.

1.3 LIMITATIONS

The methods and concepts presented in this thesis focus on freehand sketching. As such, stroke placement precision was not an important factor. This makes the implementation unsuitable for any kind of technical drawing. There are no tools to draw precise lines or to simulate a specific scale. Also, the lines themselves are approximations of the user input. If absolute accuracy is a requirement, other techniques might be more suitable. Furthermore, the concepts have been designed specifically for pen tablets. While other input devices might perform well, these have not been tested or evaluated.

1.4 ORGANIZATION

The thesis has been structured in the following manner:

CHAPTER 2 puts the thesis within the context of the research domains by giving an overview of related work. It introduces many of the concepts and technologies which will be used in the remaining chapters. Topics include aspects from both Non-Photorealistic Rendering (NPR) and user interaction.

CHAPTER 3 presents the concepts and methods designed to solve the core problems discussed in the thesis. The chapter begins with a brief description of the design philosophies upheld during the project. It continues by introducing the user interface and user interactions. The most important concepts are the novel way in which the canvas can be manipulated, new stroke interactions and scalable line art.

CHAPTER 4 gives detailed explanations of the concepts proposed in the preceding chapter. This chapter forms the technical basis of the implementation and contains the data structures and algorithms used to realize the system.

CHAPTER 5 contains a case study that illustrates the use of the concepts present in the implementation. The chapter also reports on an informal user study and the results obtained from this study. The chapter ends with a critical discussion of the implementation.

CHAPTER 6 Summarizes the thesis as a whole and touches upon possible research directions for future work.

2 FOUNDATION

In order to relate this thesis to the research domains of human-computer interaction and NPR, this chapter presents an overview of earlier work. It contains the foundations and concepts needed to understand the terms, definitions and ideas presented in the following chapters.

2.1 NON-PHOTOREALISTIC RENDERING

NPR is a very broad field where much of the research is focused on enabling a wide variety of expressive styles for digital art and illustrations. In contrast to traditional computer graphics which is focused on photorealism, NPR is inspired by artistic styles such as Chinese ink, hatching, oil painting and water colour. Areas within the field of NPR that are closely related to this thesis are *digital drawing systems*, *brush stroke models* and *automatic line drawing generation*.

2.1.1 *Interactive Drawing Systems*

The popularity of digital painting has increased greatly in the last decade or two. A large factor behind this might be the increased availability and quality of drawing tablets (Figure 1a). Previously, most were limited to using a mouse which is far from the ideal painting implement. In contrast, tablets bare a certain resemblance to traditional drawing tools in form and usage. This, combined with the benefits of digital painting have caused a shift in the illustration community from traditional to digital artwork.

There exists a wide variety of digital drawing programs, from painting simulators such as Corel Painter [Corel, 2007] to illustrative tools such as Adobe Illustrator [Adobe, 1988]. Within this variety we do not only find great differences in functionality and complexity, but also very different ways for users to interact with these programs. In some it is possible to change each line (Adobe Illustrator), in others the whole work is affected (Corel Painter), as strokes are merged into one layer. Despite their fairly recent popularity, painting systems have attracted the interest of researchers for a long time.

Early work was done by Smith who developed PAINT [2001], one of the first interactive painting programs. In PAINT, brushes are 2D shapes and users can pick colours or mix new ones. It is considered a pioneering effort in the field of digital painting. As an attempt at making the painting experience more intuitive, Smith later developed TABLE PAINT which used a stylus in combination with a tablet. This gave users the ability to change shape, size and orientation of the brush. This type of interface is still the standard and is used in our system also.

In Haeberli's seminal PAINT BY NUMBERS [1990], the colours of brush strokes painted by the user are sampled from a source image. Users can create brush strokes by clicking and dragging the mouse (Figure 3).

The size of strokes can be controlled through the keyboard or the movement speed of the mouse. The orientation of the brush strokes can be controlled in several ways, such as orienting the brush at the direction the user is moving the mouse in. The system supports different stroke shapes which lead to many different styles, such as pointillism, mosaic and painterly rendering.

Another influential effort by Haeberli is DynaDraw [1989], which took a new way of looking at digital brushes. In DynaDraw, the brush controlled by the user is modeled as a physical entity with mass. This brush is subject to acceleration and friction as the mouse exerts a pulling force on the brush with a simulated rubber band. By changing parameters such as the amount of friction or mass, different kinds of strokes can be generated. This method to vary a brush parameter with drawing speed was utilized in our system as well.

Figure 3: From left-to-right: different stages of a painting in Paint By Numbers [Haeberli, 1990]. Additional strokes reveal more of the source image, giving the user great creative control.

An ubiquitous application in the world of digital drawing is Adobe Photoshop [Adobe, 1990]. While designed for a very specific purpose, it is considered an all-round digital painting and photo manipulation system. Partly due to its heritage, Photoshop uses a raster array of pixels to store colours. Another popular pixel-based application is Corel Painter [Corel, 2007]. Painter was designed explicitly with digital painting in mind. It can simulate many types of traditional media such as oil paint, gouache and charcoal. A relatively new system which also tries to simulate traditional methods is ArtRage [AmbientDesign, 2004]. These systems dictate what is expected from a modern drawing system and were studied to develop the concepts for this thesis.

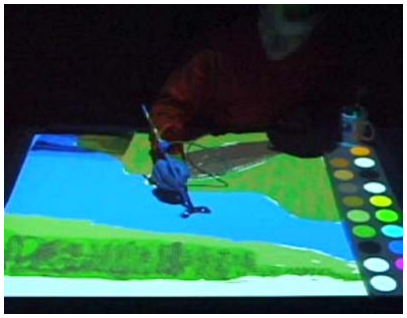
A different class of digital drawing solutions are vector-based applications. The most well-known of these are Adobe Illustrator [Adobe, 1988], Corel Draw [Corel, 2007] and Xara X [Xara]. These use geometrical primitives such as lines and polygons to represent images. These primitives are based on mathematical definitions. Because such primitives are scalable without any loss in quality (in contrast to pixel-based images), vector-based approaches are popular in the publishing community. One of the most important guidelines in designing our stroke model concept was that this type of lossless scaling would be possible.

What the professional systems have in common are large feature lists and complicated user interfaces, resulting from decades of development. While very powerful, learning to use such programs poses a daunting task for new users. The systems are also very specialized: for instance, systems designed for painting use a raster image format while systems designed for publishing use a vector approach. These approaches result in different styles. There does not seem to be a true compromise between both approaches. Aside from these commercial programs, many digital drawing systems were developed as research projects.

CavePainting [Keefe et al., 2001] takes a very different approach to digital painting. Here the user is inside a Cave Automatic Virtual Environment (CAVE) and uses physical props and gestures to paint. Props such as brushes and paint buckets (Figure 4a) are tracked in 3D space and can be used to create or modify objects in virtual space, giving the user a very immersive painting experience.

CoolPaint [Lang et al., 2003] resembles CavePainting in the sense that both utilize physical props for painting. The painting experience is recreated by having a physical brush controlling its digital counterpart. Artists can use these brushes to paint on a tabletop display which removes the spatial indirection caused by a mouse or tablet (Figure 4b). To make the experience close to real painting the brushes have six degrees-of-freedom and colour mixing is emulated with a virtual palette. The concept of using different brushes for different functions could be applied to a pen tablet input device as well.

(a)



(b)

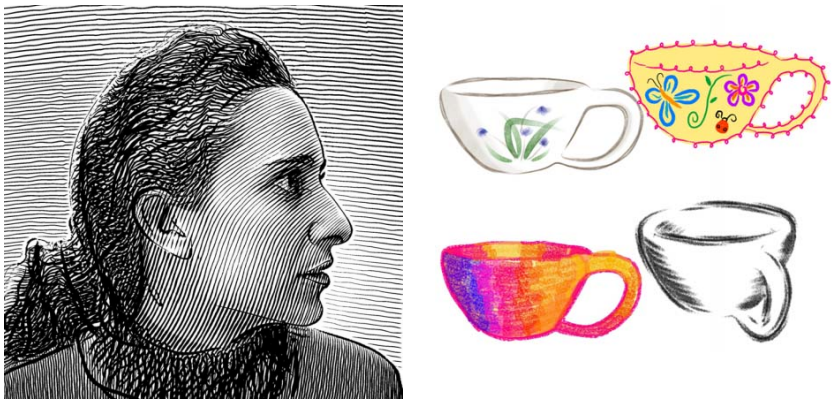
Figure 4: (a) In CavePaint [Keefe et al., 2001] the paintbucket prop creates splatter patterns. (b) The CoolPaint [Lang et al., 2003] tabletop interface in use.

Recently, Vandoren et al. [2008] created a novel painting interface called IntuPaint which uses electronic brushes with a tuft of bristles made from transparent nylon fibers. The fibers conduct infrared light through total internal reflection and cause a footprint on the canvas, which is translated into a canvas interaction. The system uses both physically based and empirical algorithms to simulate water color, gouache and impasto. The combination of the input devices with the paint simulation leads to a very traditional painting experience. The electronic brush might be an interesting research direction for new pen tablets.

ILoveSketch, created by Bae et al. [2008], tries to seamlessly integrate sketching with 3D curve creation. The system uses gestures to navigate the 3D space. A multi-stroke approach to NURBS curve generation is taken: the resulting curve is the average of the strokes put down by the user. Through similar means curves can also be connected. The system tries to emulate a sketchbook; by grabbing the corners and dragging them across the screen the user can clear the screen or save the 3D model. Results indicate that the system can be an effective tool for introducing traditional artists to 3D modeling. Some of these concepts were used as inspiration for this thesis.

The previous systems have all been designed for start-from-scratch painting. A very different category of systems are those that use an existing source as starting point. Paint By Numbers which was mentioned earlier, might also be considered an early example of this. However, Paint By Numbers only sampled the colour of the strokes from the source, and does not use the source in any other way.

Durand et al. [2001] investigated semi-automatic tonal modeling. Given a reference photo, the user can determine the placement of strokes, their tone and precision. Precision defines the amount of spatial detail of strokes, so more precision will reveal more detail from the source. The system performs in realtime so the user can see changes in the final image as they occur. Strokes are rendered using threshold textures, which allow for a wide range of media simulation, e. g. pencil drawing, charcoal sketching and copperplate engraving (Figure 5a). Strokes are rendered as Bézier curves. The visualization of the strokes and the final result are done in separate windows. This might make it more difficult to locate certain strokes as the user has to correlate between the two images. A very interesting aspect of this approach is the thresholding, which extends into a possible way in which to implement scalable stroke textures.



(a) (b)

Figure 5: Semi automatic painting using a 2D and 3D source, respectively [Durand et al., 2001; Kalnins et al., 2002].

The system of Kalnins et al. [2002] gives users the freedom to draw strokes directly onto 3D models. The model is not so much a guide for the artistic expression; but more a foundation which the artist can

change to resemble his own vision. The user has complete control over the appearance of the strokes. They provide three main categories of strokes: silhouettes and creases, decal strokes (which suggest surface features) and hatching strokes. The artist uses a pressure sensitive tablet to apply strokes to the base model. The pressure can be used to vary parameters such as width and tone. Users can save the combination of parameters that lead to a certain style for later use. Examples of different styles for the same object can be seen in Figure 5b.

A similar program which has been used in a production environment is Disney's Deep Canvas [Daniels, 1999]. This tool gives animators the ability to paint on the geometry of an animated 3D scene. The system stores the colour and placement of the strokes. By adding the hand-drawn animations as an overlay, the final scene can be constructed iteratively: the artists can keep adding paint to the scene, slowly filling in the gaps. This seamless combination of 2D and 3D could be of interest to other, 2D, drawing systems as well.

In both the commercial as academic sectors many different drawing systems can be found. The commercial systems, while powerful, suffer from great complexity. The academic systems on the other hand, exchange functionality for ease of use. They both have in common the use of Brush Stroke Models (BSM) to represent strokes.

2.1.2 *Brush Stroke Models*

In early systems, brush strokes were limited to simple lines with a static width. This limits the artist in his choice of style and artistic expression. Today, there are many different BSM. These stroke representations can vary widely, depending on the system and its requirements. In previously mentioned systems such as Photoshop [Adobe, 1990], strokes are modeled as colour values in a pixel raster. This, however, provides few possibilities to change strokes after their initial creation. In NPR systems, strokes are usually 2D geometry on top of a virtual canvas. These strokes are often designed to simulate a certain painting or drawing medium, e. g. pencil lines, charcoal, watercolour or ink. Some representations are meant as general frameworks which are flexible enough to emulate different styles.

A very influential stroke model was proposed by Hsu and Lee [1994] in their seminal paper. They present the skeletal stroke as a replacement of the constant thickness stroke. The skeletal stroke is built from an idealized 2D deformation model defined by an arbitrary path. The deformation can be applied to any image, allowing for an almost endless range of different stroke styles. The deformation is based on a localized parametric coordinate system along the stroke path, while the stroke itself is defined by a backbone and thickness. Skeletal strokes can be evaluated in realtime, and so are suitable for interactive painting systems. Examples of skeletal strokes can be seen in Figure 6. A variation of the skeletal stroke would be to map the image to a polygon with texture coordinates, instead of directly deforming the image. This is an approach used often in drawing systems. The way in which strokes are rendered in our system is loosely based on the skeletal concept as well.

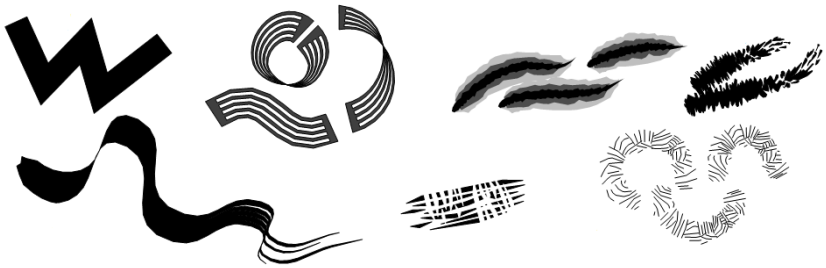


Figure 6: Skeletal strokes with different textures and properties. Anything from wood-cut to an ordinary flat-nib pen can be recreated [Hsu and Lee, 1994].

Seah et al. [2005] represent strokes as mathematical curves. In their approach, the center line of a stroke is encoded as a disk B-spline. The control points of this spline can be given attributes such as a radius (hence the disk) or colour. Strokes can be easily manipulated by moving control points as the 2D stroke area between them is implicit from the spline definition. The result is a very storage-efficient, flexible way to represent brush strokes which is suitable for a range of different styles, for example calligraphy. However, they do not handle the issue of processing the user input to extract a sparse representation of the stroke. The internal representation of strokes in our systems was directly inspired by this approach.

In an earlier work by Su et al. [2002] a similar method is used, also with B-spline curves. They focus more on the aesthetic aspects of the stroke however. By evaluating certain functions along the lengths between control points, a wide range of visual styles can be achieved. To illustrate the method, they create a close analogue of Chinese ink by varying ink density along the line. This approach to calculating the appearance makes it very suitable in situations where the artwork must be available at large scales, as it does not lose quality.

Originating from a very different direction are curve analogies, proposed by Hertzmann et al. [2002] The method is inspired by algorithms from image texture synthesis: given an example curve, artists can draw other lines in the same style; the example is mapped to the new curve to make them look alike. This gives artists a very intuitive way to define a style and easily maintain it consistently throughout an illustration. The process involves sampling from the example curve so the neighbourhood around each point in the new curve resembles some neighbourhood from the example. This concept could be an interesting extension for a drawing system.

Murakami et al. [2005] present an algorithm that creates realistic pastel and charcoal strokes. They represent strokes with Catmull-Rom splines and use the calculated maximum pressure and height fields to calculate the pigment deposition on the canvas. The height fields are generated from twelve real paper textures, lit at different angles. The strokes are rendered as triangle strips. The results are very convincing, as can be seen in Figure 7. The difference in regard to the other stroke models is how they use the path of the stroke to calculate where the maximum

Figure 7: A computer generated pastel drawing of two mammals [Murakami et al., 2005].

pressure applied by the pencil would be.

While stroke models vary widely in appearance and complexity, the underlying representation is often very similar. Most, including our own, use the fundamentals proposed by Hsu and Lee [1994] in some way. Splines are a very effective way to interpolate segments of a stroke, and play an important role in many models. One particular area where a stroke model is often used are systems that perform automatic generation of line drawings.

2.1.3 *(Semi) Automatic Generation of Line Drawings*

While direct user input is one way to determine the placement of strokes, another is to automate this process. A popular approach to this is Stroke Based Rendering (SBR): a, possibly automatic, approach to creating non-photorealistic images by placing discrete elements such as paint strokes. One way to determine how the strokes are positioned and what their parameters are is to use a 2D or 3D source. This does not mean the user cannot influence the stroke placement in some way. Rather, most SBR systems depend on user interaction to produce a visually pleasing result. Many different styles have been implemented as automatic systems, simulating pencil strokes, pen-and-ink, watercolour, oil paint and mosaic. The rendering of line drawings has seen significant attention. Another popular topic is creating a stylized rendition to better convey relevant surface information of 3D objects. A brush stroke model is often an integral part of these systems. While SBR might be a popular way to automatically generate line drawing imagery, it is not the only one. Another well known approach is to directly extract the lines (with image or object space techniques) and render these, possibly after some modification.

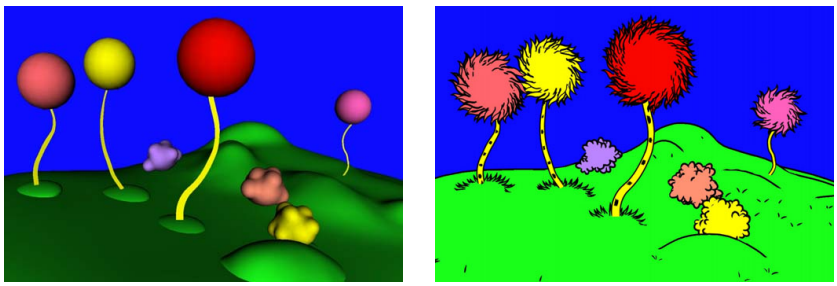
Hertzmann [1998] describes a framework for painterly rendering. Brush strokes are modeled as splines whose colour is picked from a source image. The result is built from multiple layers of splines, each successive

layer having a smaller brush size. A number of styles can be achieved, e. g. impressionism and expressionism. Here, the brush strokes are B-splines, a representation commonly used in drawing systems, whose direction and placement is calculated from the gradient of the image. A drawback of this approach is that the user has very limited control over the flow of the strokes, aside from a few global parameters he can define. Also, each stroke can only have a constant colour.

In an early attempt to better convey information about an object, Dooley and Cohen [1999] use information from the hidden surface removal process to determine line style, width and transparency. The lines can be solid or dashed in appearance. Determination of these parameters is done from user specifications and inferences drawn from artistic knowledge and illustration principles.

In a similar endeavor, Elber [1995] uses the wireframes (polygon edges) of 3D objects to convey simple information about their shape. With varying line width for depth cueing and by trimming hidden edges at intersection points, a clear and convincing representation of the object is created.

Kowalski et al. [1999] present an algorithm for drawing stylized, procedural fur and grass strokes on 3D images. By dividing the screen into patches called graftals and assigning different textures to these graftals based on the underlying geometry, the complexity of the strokes can be represented implicitly in the geometry. Using this method, they recreate the distinct illustrative styles of Theodore Geisel (Dr. Seuss) and Geoffrey Hayes (Figure 8). Defining strokes through textured patches is an approach which was also considered for our system.

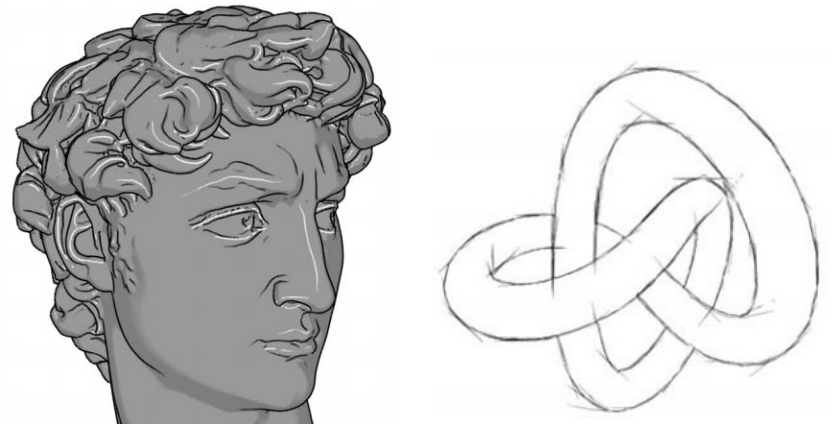


(a) (b)

Figure 8: A typical situation in SBR: (a) shows the source, in this case a 3D model of a landscape. (b) Shows the resulting strokes generating using the source as a guide [Kowalski et al., 1999].

Lee et al. [2007] created an GPU-based algorithm for rendering 3D models as line drawings. They see a line drawing as an abstraction of a shaded image. Such an image is used as a basis and lines are drawn along tone boundaries or thin dark areas. Highlights are also drawn along thin light areas. This creates a mix of feature lines and suggestive contours, which convey the essential features of the shading. Their results can be convincing and, thus, an effective way to render sketchy lines. The stroke model here is similar to that used by raster based systems: they are drawn together on a single layer, in this case a texture.

The strokes cannot be altered dynamically as the stroke parameters cannot be retrieved from the layer. DeCarlo and Rusinkiewicz [2007] present a similar method. They put more focus on sparse white highlight lines (Figure 9a). They provide an object-space method to extract these lines. They also look at several stylization options in rendering these lines. Some of these stylizations were considered for our system.



(a) (b)

Figure 9: Automated sketching. (a) An automatically generated sketch of David with white highlights [DeCarlo and Rusinkiewicz, 2007]. (b) Loose, sketchy rendering of contours [Lewis et al., 2005].

In a slightly different direction, Barla et al. [2005] present an approach to simplify line drawings. From the original set of lines a smaller set is created which represents the geometry of the original. Their approach has two phases: identification of line clusters which can be merged within a certain error and processing these clusters to create new lines from them. The behavior of the second phase depends on the type of application. The technique is effective at reducing visual clutter, reducing line drawings to their essential lines. In a drawing system this might be an effective way to keep the number of redundant lines to a minimum. The artist would add to existing lines to increase their weight instead of drawing additional lines along the same path. However, Barla et al. simplify finished drawings with the method, not work that is still being drawn. It is uncertain if the approach would work under such conditions.

Lewis et al. [2005] address the problem that rendering contours as a single, continuous stroke is not a very realistic portrait of a sketch drawn by an artist. They seek to emulate the character of a sketching stroke by approximating contours with several shorter strokes. They treat the stroke segmentation as an instance of a clustering problem. The proximity and orientation of curve samples are compared and similar samples are grouped into strokes. The result is recognizable as the type of short, quick lines most people produce when sketching (Figure 9b). The stroke model used is very simple; splines modeled as lines with a constant thickness and adjustable opacity. This shows that convincing results can be achieved with a simple model.

Grabli et al. [2004] present a method which gives the user greater control over the style of the final result. Their model represents drawing as a sequential process; decomposing a drawing into individual operators for the selection, chaining, splitting and ordering of lines. These atomic operations can be defined and ordered by the user using the Python programming language. The visual result can vary greatly depending on which operations are used, and in which order. The stroke model used is a variant of the skeletal stroke [Hsu and Lee, 1994]. While the idea of using a sequential system is not entirely new (3D rendering systems use similar systems to define materials), its application to line drawing is. Although interactive development without recompilation is possible, programming a style is not the most intuitive way to define one. This was taken into account while designing the concepts presented in this thesis.

Given a source and some guidance from the user, semi automatic systems can generate interesting imagery. Many different artistic styles have been automated with these systems. While the emphasis is placed on automatic image creation, the way the user can interact with these systems is still very important.

2.2 USER INTERACTION

In the early days of computer science, handling user interaction was often regarded as an afterthought. However, a program that cannot be used is as good as no program at all. User Interface (UI) design is a very active research area and User-Centered Design (UCD) has become a popular design philosophy. For drawing systems, the interface and functionality are very strongly intertwined; the system must respond appropriately to the movements of the artist. The type of interaction can greatly affect the artist's ability to be productive and accurate. There are many ways for users to interact with such systems, as will be illustrated.

2.2.1 *Interaction With Large Displays*

Large displays hold many attractions and possibilities. Their size makes them a good option for sharing information between users who are co-located. The greatest advantages of a large display in digital painting are its ability to show fine detail such as lines and the large work area they provide. Interesting is the many ways in which users can interact with the screen; an area in which much research has been done. There are a few types of interaction that are particularly popular.

Interaction By Touch

One of the most natural and effective ways to interact is through direct touch. Usually this is done through a pen or using the fingers to touch the screen directly. By themselves these are limited in how they allow users to interact. Often they are combined with gestures or postures to make them more powerful tools. Both Wu and Balakrishnan [2003] and Grubert et al. [2008] present whole hand postures which extend the number of possible interactions by giving each gesture its own meaning. Postures used by Grubert et al. included one finger, two fingers, flat hand and fist (Figure 10). Wu and Balakrishnan used

vertical and horizontal hand postures. These postures are essentially used to create a different cursor which can be moved over the screen. Different from whole hand postures are the time dependent gestures. Wu and Balakrishnan explored a number of these, e. g. tap, double tap, flick and catch. These gestures are reported to be easy to grasp and use, regardless of type. While the gestures add an intuitive, modeless method to interact with a system, the whole hand gestures give an interesting way to change the type of interaction the user is performing without pushing any buttons. Many of the interface concepts presented in this thesis were inspired by this gesture-based approach.

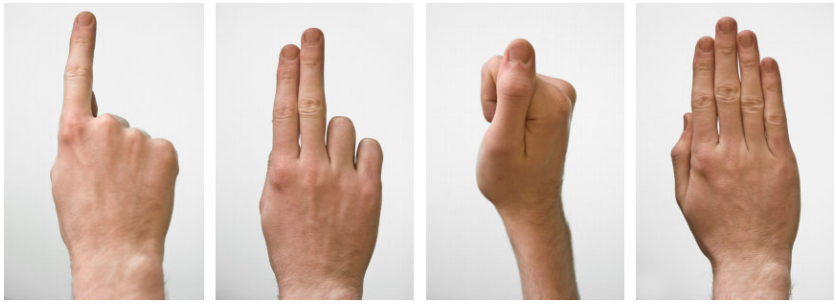


Figure 10: Different types of whole hand gestures used by Grubert et al. [2008]. The gesture determines what kind of action the user is performing.

Another way to increase the functionality of touch displays in a natural feeling way is to add more elaborate command gestures. Drawing these gestures on screen calls a certain action. Command gestures can be arbitrary shapes, such as circles and squares, or mnemonic, so that the shape corresponds to the action performed after drawing it. For example, drawing a 'c' will invoke a copy command. An early attempt at using such gestures was done by Rubine, who describes GRANDMA [1991], a system that helps users specify custom gestures. Bau and Mackay present OCTOPOCUS [2008], which helps users learn and execute gesture commands. The system attempts this by using dynamic guides. An example of such a guide is the prefix, which shows the part of the gesture that still needs to be drawn and so directs the user. This could be generalized into a method that helps a user learn the controls of a system, while not being interruptive. In a drawing system it might even be used to suggest shapes the user can 'trace over'. This would help the user draw difficult shapes such as perfect circles, while retaining the users own drawing style.

A possible issue that can arise with direct-touch interaction is the accuracy. The fingertip of an average adult will obscure the exact location of the cursor. Also, because of the surface area that is touching the display, the cursor might not be where the user expects it. This problem, sometimes referred to as the 'fat finger problem', might not make direct-touch interaction suitable for precise drawing.

Bimanual Input

The way in which we use our two hands can be split into two categories. The first is unimanual, such as writing or stirring a spoon. The second is bimanual, which includes actions like tying shoe-laces and weight

lifting a barbell. Tying shoe-laces is considered an asymmetric bimanual task while weightlifting a barbell is a symmetric task. Research into bimanual input has been carried out for some time, but has recently gained in popularity. Bimanual input decreases the disruptiveness of using the interface while painting. One of the earliest publications was by Buxton and Myers [1986], who found benefits over one-handed methods in using the non-dominant hand for tasks like scrolling, while using the dominant hand for precise selection. Bimanual interaction could prove to be useful in digital drawing.

Balakrishnan and Hinckley [2000] focused on symmetric bimanual interaction in particular. They found that a lack of visual integration between the task and the hands resulted in greatly increased asymmetry between the movement of both hands. Task difficulty and attentional demands did not affect the level of symmetry. They conclude that bimanual interaction is very suitable for navigational tasks. Also, in symmetric tasks, the human motor system does not seem to devote more resources to the dominant hand when attention is divided.

Forlines et al. [2007] investigate the differences between using direct-touch and mouse input for both unimanual and bimanual interaction on tabletop displays. Their research indicates that direct-touch is favorable over two mice regarding bimanual input. In unimanual situations, the mouse performed much better in terms of speed and accuracy. When external factors such as fatigue and spatial memory are included, direct touch might also be considered for such tasks.

2.2.2 *Peripheral Devices for Drawing*

The most direct and important type of user interaction is the input device. Without the mouse and keyboard there would be no interaction with modern computers. While these two are by far the most prominent and ubiquitous, they are not the most suitable for digital painting. Over the years, many attempts have been made to create a better input devices for such tasks. Other devices, while at first glance meant for a different task, might also be suitable for painting.

The most well-known and successful digital painting input device is the Wacom tablet series [Wacom]. The Wacom tablets use stylus pens that are both cordless and without battery (Figure 1a). Communication between the stylus and the tablet is done using a patented electromagnetic resonance technology. While not an accurate simulation of the actual pen and paper dynamics, becoming articulate with the tablet takes only little practice. Thanks to the ease of use and their wide availability, Wacom tablets have been the graphics industry standard for well over a decade. Similar technology is also used in modern digital white boards, which function along the same lines. These pen and tablet systems should be well supported by any drawing system, as they are the corner stone in digital painting interaction.

A recent addition to the group of widely available input devices is the Wiimote [Nintendo, 2007] (Figure 11a). While originally meant as the control device for the Wii gaming console, its novel interactive properties have caused it to find its way into academic research as well.

Schlömer et al. [2008] use the Wiimote's acceleration sensor for gesture recognition, while Gallo et al. [2008] use the device to interact with volumetric medical data. The Wiimote could possibly have some merit as a remote painting device. Its rotation and angular sensitivity might be usable as brush parameters. The greatest concern is how accurate the user can be with the wiimote. Still, it could prove very effective at painting in certain styles where sweeping motions work well.

(a) (b)

Figure 11: (a) The Wii controller, commonly called the Wiimote [Nintendo, 2007]. (b) Using a tape based input device to manipulate a 3D surface [Grossman et al., 2003].

Another well known remote input device is the dataglove. When worn, the glove allows the user to manipulate an environment, which is usually in 3D. Wan et al. [2004] present an approach for the virtual grasping of 3D objects using a dataglove. The method is used in conjunction with a virtual hand model to increase realism. The CavePainting system [Keefe et al., 2001] discussed earlier also uses such a glove. CavePainting shows that the dataglove is a viable input device for painting, even though free-handed interaction implicates a certain inaccuracy.

A very unique input device was used in the work of Grossman et al. [2003]. The device is essentially a line of tape (Figure 11b). It can be bent into many shapes by the user. The tape contains fiber optic sensors which provide twist and bend information. This information can be used to virtually reconstruct its shape. The tape can be used to create and manipulate 2D and 3D curves. Possible applications include 2D drawing and specifying surfaces in 3D. Its application in a sketching system are debatable; the sweeping pencil movement is very important for sketching, but is something the tape cannot reproduce.

There are many interesting ways to interact with drawing systems. Touch-based interaction, both unimanual and bimanual, is a very attractive possibility for large displays. There are also a number of peripheral devices which could be interesting for painting applications.

2.3 SUMMARY

Painting systems are commonplace. Most of these systems have an over-abundance of functionality. They often suffer from complex interfaces that require significant effort to master. They also tend to be

highly specialized for a certain task. On the other side of the spectrum are the programs that focus on an intuitive, immersive user interaction experience, usually at the cost of functionality.

Painting systems often use models to describe the appearance of strokes. Many different brush stroke models have been presented over the years. A popular approach is to use curves to interpolate between known points of the stroke. Independent from which method is used to generate the stroke, often the final output is a variation of the skeletal stroke concept: a base path with a variable thickness, rendered as a triangle strip, usually with some type of texture applied to it.

These stroke models are used in many different types of drawing systems. Where start-from-scratch systems are meant to give the user full control, automatic systems do the opposite: given a source image or scene as a guide, the system automatically renders a NPR image. Often the user is given a fair amount of freedom in specifying the style of the output. For semi automatic systems user input is very important. The system helps the user produce a certain result, but does not take control.

The way these systems are presented to the user is important. Large displays are an interesting option for digital painting. Sizes are similar to an average painting canvas and the high resolutions allow for fine detail to be painted. There are many distinct ways to interact with large displays, for example touch-based interaction. Some are more suitable for painting than others. In a more general sense, there exist many interesting input devices to paint with. While some are very specific in their application, others open up a wide range of interaction possibilities.

What this thesis will address is a specific type of drawing that no current system was specifically designed for: sketching. Current systems are missing a flexible, intuitive way to manipulate lines, while at the same time keeping a natural feeling sketching interaction. The interfaces of most systems impede the drawing process. They are overly complex and use many mode changes to switch functionality. With sketching, we want to focus on the work itself. The interface should be simple, effective, intuitive and unobtrusive. Sketching differs from painting in that it is a different phase in the creative process. Sketches are an artist's tool to help form ideas or try out different concepts. A sketching system should reflect this and give the user a robust, intuitive way to sketch and change sketches.

The following chapter will explain the core concepts and design philosophies of the thesis.

3 NEW IDEAS FOR FREEHAND SKETCHING

This chapter addresses the problems outlined in the previous chapters. Brief, conceptual explanations are given that show which methods are present in the implementation and what their benefits are. The design of the interface is also discussed in this chapter. Most of these concepts and methods have resulted from an informal observational study and informal communication with artists.

3.1 DESIGN PHILOSOPHY

First we discuss the concepts that remained a constant factor in the development of all the functionality present in the system. These concepts form an integral part of the philosophy behind the project. It must be noted that the project was designed with a pen tablet in mind as primary input device.

3.1.1 *Aesthetics*

An often neglected aspect of research systems is the visual design. While it has no direct influence on the functionality of the program, it does affect the user. A better interface can improve the performance of users and their acceptance of a new program [Chatty et al., 2004]. For this reason the system was designed with a pleasing and consistent look and feel. The goal was an interface which does not distract, but still invites the artist to use the system. Furthermore, we wanted to set the system apart from other drawing systems. Inspiration for the interface design was drawn from ArtRage [AmbientDesign, 2004] and ILoveSketch [Bae et al., 2008]. Care was taken to make sure the quality of graphical elements is constant regardless of screen size. To that end the native size of the buttons, for instance, is quite large. The interface can be seen in Figure 12.

3.1.2 *An Effective Minimal Interface*

Keeping the interface bare and simple results in the artist spending less time navigating the interface and more time drawing. Finding a specific action in the vast menus and option lists most drawing systems use is distracting. We wanted an interface that was still effective but used simpler and more intuitive controls. Also, it was important not to sacrifice existing functionality for the sake of a simpler interface, so great care was taken in choosing interface methods that suited the system. Still, the system had to be recognizable for people who have had experience with menu-driven products, such as Corel Painter [Corel, 2007]. We tried to incorporate established interface techniques which have proven very effective.

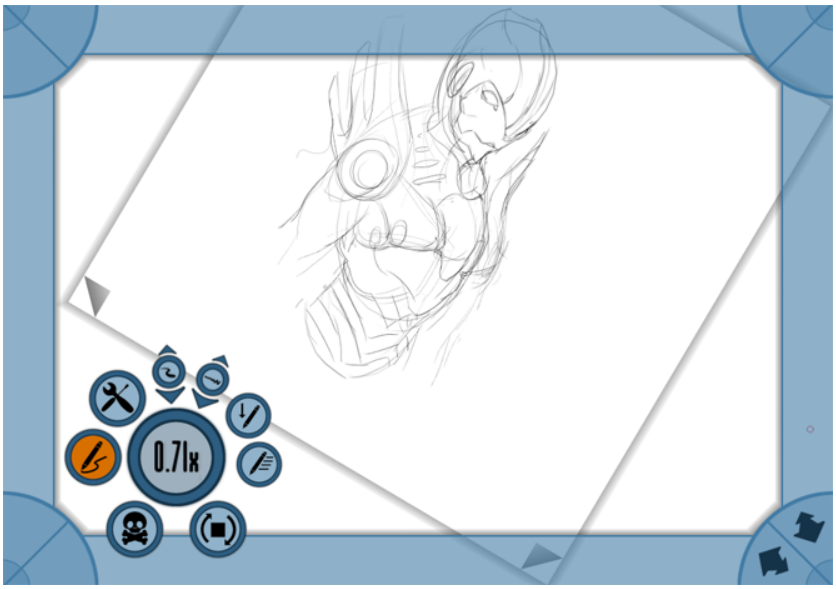


Figure 12: The GUI used in the sketching system.

3.1.3 *Limiting Mode Changes*

Mode changes are interface interactions whose sole purpose is to put the system into a certain operating mode. Such interactions can be anything from pushing a menu button to using a keyboard shortcut. Simple examples of operating modes would be drawing, erasing and selecting. Forcing the artist to click menu buttons to activate these modes is cumbersome and distracting. Therefore, an important design philosophy was to limit the number of mode changes in the system as much as possible, and use other means to change the operation the artist can perform.

3.2 USING THE CANVAS AS A TOOL

The feature which most drawing systems lack is the usage of the canvas itself. The canvas is a great tool which can assist the artist in different ways: the orientation of the canvas can make certain lines easier to draw, while the canvas bounds give a general sense of direction and helps establish the frame in which the artist is working. Being able to zoom in makes it easier to draw details, while zooming out helps the artist get a sense of the composition, much like a painter who takes a few steps back to observe his work. We attempt to recreate this functionality by presenting a minimalistic interface for intuitive canvas manipulation.

3.2.1 *Manipulating the Canvas through Gestures*

In drawing systems, methods by which the canvas can be manipulated are often complicated or non-existent. To access these interactions, the artist must use specific hotkey combinations or navigate menus. Once accessed, the interaction still needs to be performed, usually in a way similar to selection transformations. In some systems, such as Adobe Photoshop [Adobe, 1990], the selection transformation is the only way in which drawings can be oriented. Translating the canvas is usually

done through scroll bars, which are slow, as the arrow buttons only translate by a small amount on each click. Also, finding and dragging the scroll indicator is distracting. We attempt to solve these problems with a novel approach to canvas interaction.

Some attempts to get more use out of the canvas have been made in the past. Most notably, Fitzmaurice et al. [1999] proposed a way to rotate the canvas. More recently, the ILoveSketch system [Bae et al., 2008] implements several gestures with which the artist can change the canvas view without using any mode changes or extra buttons outside those of the pen tablet. Because of their modeless nature, low learning curve [Rubine, 1991; Bau and Mackay, 2008] and intuitive feel, a gesture based approach was selected for this system also.

Gestures form a very important part of the system. The system uses simple, straight line gestures. While drawing straight lines might not conform to the traditional idea of (complex) gestures, we use the term to avoid confusion, as 'drawing lines' is used to describe a drawing interaction as well. In the context of this thesis a gesture is simply an interface interaction which is both modeless and location sensitive. Many of the interactions can be done exclusively through gestures. The most prominent part of the system that is affected by gestures is the canvas. The edges of the screen, which are covered with a blue coloured frame, are sensitive to gestures which manipulate the positioning of the canvas. These frame gestures give the artist control over three canvas interactions: *translation*, *rotation* and *magnification*.

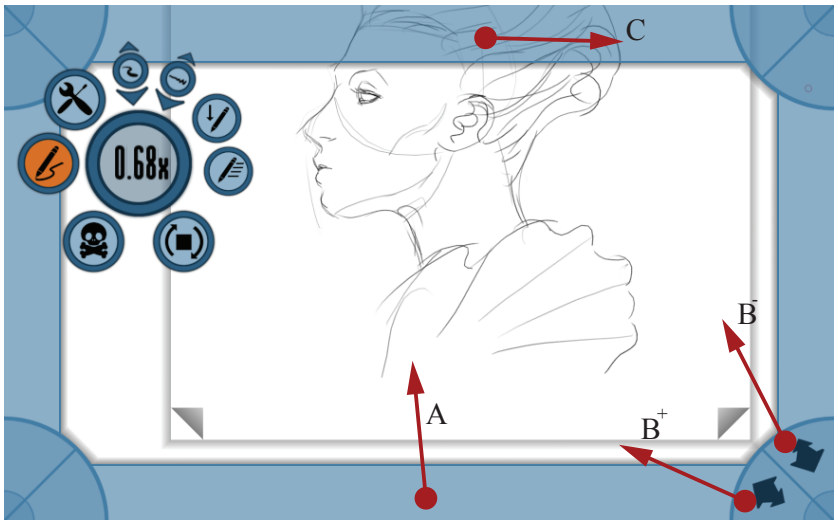


Figure 13: Canvas manipulation. The arrows depict the click and drag motions for the different frame gestures, where A is *translation*, B is *magnification* and C is *rotation*.

Translation, or moving, uses a straight line gesture. It can be done by starting an interaction on the frame and dragging towards the screen center. When the gesture line is long enough, it is recognized as a translation interaction. The artist can now freely move the canvas in the dragging direction, giving the artist a feeling that he is grabbing and pulling the canvas. The gesture is quite forgiving, so the motion does not need to be exactly towards the center. Once the translation

interaction has been established, the artist can move the canvas towards any location. *Rotation* can be activated by dragging the cursor parallel to the frame edge. Again, once the interaction has been established after drawing a line which is long enough, the artist is not limited to the frame to perform it. Finally, *magnification* can be performed from the frame corners. These corners are divided in two: dragging from one half toward the screen center zooms in, while dragging from the other zooms out: one is the inverse of the other. All four screen corners can be used in this way. While theoretically one zone would have been sufficient, e. g. zooming in by dragging away from the center, the screen edge limits the freedom for zooming in quickly. Thus, zooming was split into two interactions which both have the same amount of freedom. The gestures have been depicted as red arrows in Figure 13.

The frame gestures provide a modeless way to interact with the canvas. It tries not to take the canvas metaphor too far, such as having to perform the actions on the canvas itself. Because the frame is always visible, the gestures are always available. The screen edges were chosen for this type of interaction because they are easy to reach, even with little accuracy [Appert et al., 2008]. While using screen edges for interaction is not new, it is rarely seen in practice. Although, consequently, in the current interface the frame does take up a fair amount of space, this is outweighed by the benefits the artist receives in terms of interaction. The frame gestures are one of the features that sets the system apart from others. As mentioned, interaction with the canvas is usually cumbersome and detracts from the drawing itself, as it requires interaction with menus or scrollbars. These require fairly precise navigation from the artist to operate. The frame gestures, in contrast, can be used easily and quickly. No precise navigation to a certain button or slider is needed, the artist must only move the pen to the border. The areas on which a certain gesture can be used are large and, therefore, hard to miss.

3.2.2 *A Sketchbook Analogy for Saving*

Saving and loading drawings is a somewhat technical process. The artist needs knowledge of the file system and file formats. This is an assumption that should not be made, as many (older) artists who are now turning to digital art have little experience with computers. Also, the acts of loading and saving themselves require a fair amount of interface navigation and, thus, are time consuming.

We preferred an approach to saving and loading that was both quick and transparent. It should not expose any unnecessary technical details to the artist. The chosen approach was inspired by Bae et al. [2008], who uses a very elegant method which closely resembles a real-world sketchbook. The idea used in the system is to have two page marks on the bottom of the canvas, shown in Figure 14. Pressing the left page mark will load the previous sketch and pressing the right page mark will load the next sketch, or create a new sketch if there is no next one. The current sketch is saved when the artist changes sketches. This way the artist maintains a real sketchbook, with pages he or she can flip through. One obvious downside to the approach is that the artist has no control over the location the sketches are saved at. A possible solution

would be to allow the artist to choose such a location, or add a regular save method for those who prefer that.

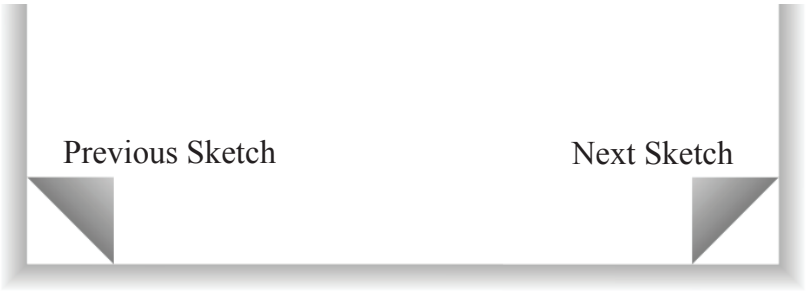


Figure 14: Sketchbook analogy. The saving 'widgets' have been made part of the canvas.

3.3 REMAINING INTERFACE ASPECTS

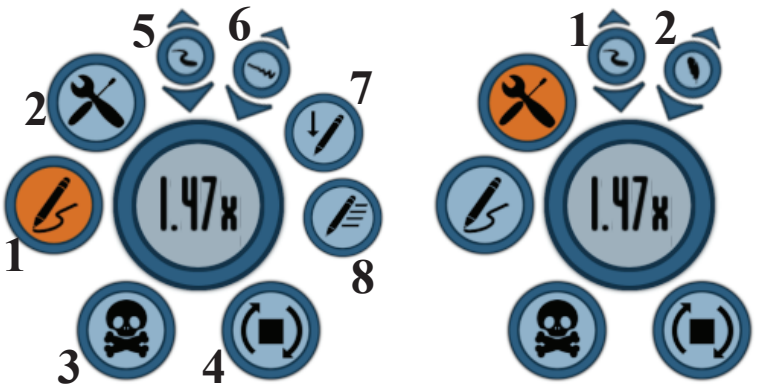
Aside from the canvas, there are several other important interface parts.

3.3.1 *Menu System and Interaction Modes*

Many drawing systems use extensive arrays of button and drop-down menu's to provide access to each interaction. This can be overwhelming for first-time users and forms a steep initial learning-curve. By following the design guidelines we have attempted to reduce this complexity in our system.

Like the rest of the interface, the main design focus of the menu system was minimalism. It needed to be simple and small. The menu is made up of a round status display around which the buttons are placed. While the bigger buttons are permanently visible, the smaller buttons change depending on the current interaction mode. The middle circle contains information concerning the parameter currently being manipulated by the artist. This can be anything from brush size to canvas magnification. The circular design was chosen because it bears a fleeting resemblance to the pallet used by a painter. Also, it keeps all the interface buttons in a compact shape. Most drawing systems use long, square menus that are difficult to place due to their size. Furthermore, it is an aesthetically pleasing way to display a menu.

The large buttons provide a basic interface with the system. The default mode of the system is the *drawing mode*. In this mode the artist can draw strokes on the canvas with the pen-tip and erase them with the eraser-tip of the tablet pen. Other functionality which is available in this mode include stroke selection and of course frame gestures. The *edit mode* gives the artist a very different set of controls. In this mode, the pen tip controls the move brush, while the eraser-tip activates the move stroke. Both of these interaction types will be explained later. The different menu modes can be seen in Figure 15. The remaining two large buttons affect the canvas: the *clear* button simply clears the canvas, erasing all the strokes. The *view-reset* puts the canvas view back into its default orientation and magnification. It can be considered



(a) (b)

Figure 15: The menu system. (a) shows the menu displayed when in drawing mode. The buttons are: **1.** Draw mode **2.** Edit mode **3.** Clear canvas **4.** Reset canvas view **5.** Brush size **6.** Smoothness **7.** Pressure variable **8.** Speed variable. (b) shows the edit menu. Here **1** is the brush size and **2** is the brush softness

as a short-cut for a series of frame gestures. There are two types of property buttons for the draw and edit modes: regular press buttons and drag buttons. The drag buttons increase or decrease the value of their respective parameter by pressing the button and then dragging it towards the direction indicated by the button. This eliminates the need for value drag bars, and thus keeps the interface smaller.

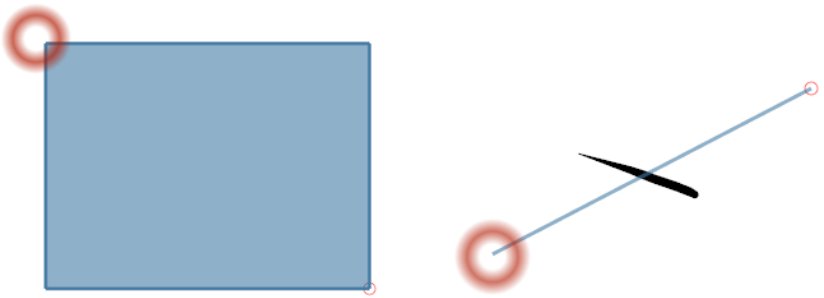
3.3.2 *Tap Interaction*

To further reduce the number of interface buttons (and mode changes), the system also makes use of tap gestures. The tap, or click, is one of the oldest gestures found in user interfaces. The double-click, for instance, has been an integral part of many operating systems for decades. The system uses the tap gesture in several ways: tapping the canvas in close proximity to a line will select that line, while tapping an empty part of the canvas will create a tap marker. Tapping can be done with both the pen and eraser, and the tap fades out over time as shown in Figure 16.

Figure 16: A fading tapmarker.

The tap marker is used for two canvas interactions: *multi-select* and *erase-line*. The tap marker fades out over time and is only active for a short time. This prevents the marker from interfering with the drawing process too much. A *multi-select* can be performed by clicking the tap marker with the pen and dragging. This creates a rectangular border, much like the multi-select in most applications. Completing the drag will select all the strokes that intersect the rectangle. *Erase-line* is done by dragging the eraser tip from the marker. This produces a single line which runs from the marker to the pen cursor. All the strokes that

are intersected by this line will be erased when the drag is completed. These two interactions are shown in Figure 17.



(a) (b)

Figure 17: The tap interactions. (a) shows the *multi-select* while (b) shows the *erase-line*.

A pen tablet has a very limited amount of differentiable inputs. Using tap interaction in this novel way is a solution for dealing with these limited input options. The system does not use the keyboard for any of the interactions, so the artist does not need to go back to a keyboard, thus maintaining the 'one input' metaphor. The tap marker creates a new context in which these inputs can be used, in this case selection and erasing.

3.4 INTERACTING WITH STROKES

Creating and changing strokes is one of the central themes of the project. We desired an approach that would remain faithful to free hand sketching but would still be able to incorporate the new interactions and scalability.

Stroke Transformation

In modern drawing systems, (stroke) selections can often only perform one or two types of transformations at the same time with a selection frame. In some cases, additional transformations can be accessed through very precise positioning or hotkeys. Often, however, the artist will need to change the selection mode through interface interaction to use other transformations. Our goal was to provide easy and quick access to all major transformation types through one selection frame, without any type of mode changing.

When one or more strokes have been selected, by clicking them or using the multi select, a frame will appear around them. The frame has blue edges and circular borders. One corner will have three buttons to either cancel, erase or define the selection as a move stroke. All of the standard transformations can be performed on the selection: *translation*, *rotation* and *scaling*. The gestures to control these operations are very similar to those used with the screen frame. *Translation* is performed by pressing down on a selection edge and dragging it outwards. *Scaling* is done by dragging the corners of the selection from or to the selection center. For *rotation* the artist has two choices: dragging parallel to a selection edge or towards the center and dragging the selection corners

in a direction perpendicular to the scaling movement. These gestures are marked in Figure 18.

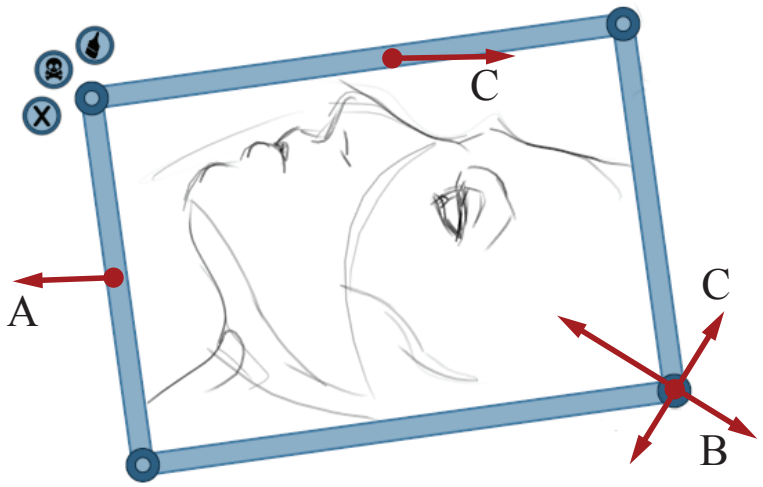


Figure 18: Stroke transformation. Again, the arrows depict the click and drag motions for the different frame gestures, where A is *translation*, B is *scaling* and C is *rotation*.

The transformation concept is very well suited for gesture-based interaction. The choice to use interactions very similar to the canvas interactions was made to keep the interface as uniform as possible. Once the artist learns one set of controls, he or she can apply this to different parts of the system. It is important to make the user aware of the places where a certain control set is valid. In this case, a visual clue is given: both the screen frame and the selection border are blue and have circular corners. Still, in many regards the selection border is similar to those used in most drawing systems. This was done intentionally; those selection mechanisms have been tested for decades and have proven very effective. What most systems do not do, however, is combine three operations into one border where all three can be used without changing modes. This is what makes the gesture-based interaction so effective.

3.4.1 *Drawing Strokes*

Drawing digitally is something which can already be done very well. A pressure sensitive pen tablet combined with the brush model from any recent drawing system can produce very convincing results, regardless of the style and medium the artist wishes to reproduce. We do not see any reason to change this already very effective and established interaction. Therefore, putting down strokes on the canvas is done in the same way as any other system: simply touch the tablet with the pen and strokes will be drawn. We propose to use both pressure and drawing speed to determine the appearance of the stroke. These can be used to vary the size or opacity of a stroke. This gives the artist some flexibility in choosing a style and to work with a brush that feels comfortable.

3.4.2 *Intuitive Editing of Strokes*

One of the focus points of this thesis and one of the most important stroke interactions is stroke shape editing. In drawing systems, methods for editing the shape of strokes are cumbersome or limited. In vector drawing systems, for instance, it requires the artist to manipulate a spline curve, which is not a very intuitive process. In raster programs it can not be done at all as strokes are 'baked into' the same layer. We hope to improve on this with several novel approaches to stroke editing.

Move Brush

The simplest way to edit strokes is through the usage of a move brush. The move brush is available through the pen tip of the stylus while in *edit mode*. Generally, a move brush works much like a regular brush. It is a radial brush that can be dragged over the canvas to apply it (Figure 19a). In function, the move brush is very different; when dragged over a stroke it displaces the stroke interval it is in contact with in the direction the brush is moving. This results in a very easy to use, natural feeling way to alter the local shape of a stroke. The brush will move all the strokes it comes in contact with. This is a necessity because, while sketching, an artist will often trace the same line several times. Reasons for this can be adding more value to a line or to better approximate the curve the artist wanted to draw.

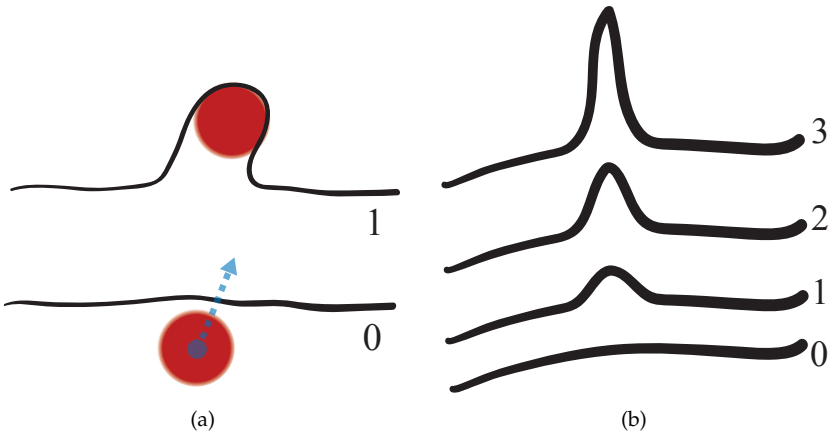


Figure 19: The move brush. (a) shows how the movebrush can change the stroke shape with one action. In (b) the effect of the 'warp' tool included in some vector programs is shown, after several uses.

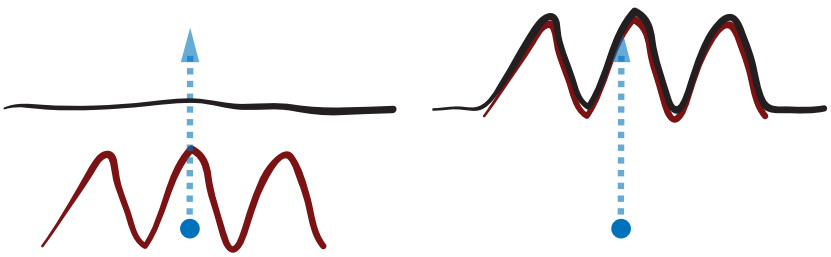
The move brush can be used in several ways. The most obvious one is to adjust lines to better conform with the intent of the artist. Instead of having to erase the line and redraw it, the artist can adjust the line where it deviates. This can be desirable when erasing and redrawing incurs a high cost for the artist. A situation where this might hold true is when the error in the stroke is small: most of the stroke has been drawn accurately and redrawing it might give a inferior overall result.

Another use for the move brush is more creative. Because the displacement does not preserve the original stroke length, the move brush can

be used to change simple strokes into more complex ones. The process is more or less the reverse of sculpting: instead of taking away, one keeps adding to the shape. Other drawing systems, Illustrator [Adobe, 1988] in particular, also have tools to 'warp' the shape of a line or stroke (Figure 19b). However, these tools are less general (only one type of brush) and try to preserve the line length, which limits the amount of editing which can be done. This causes such tools to feel less responsive.

Move Stroke

Move strokes can be seen as a generalization of the move brush. Whereas the move brush is a simple radial brush, the move stroke can take on the shape of any stroke defined by the artist. A move stroke is created by selecting an existing stroke and pressing the 'set as brush' button. The move stroke can be dragged across the canvas by using the eraser tip of the stylus while in *edit mode*. The move strokes behaves like the move brush, but with one notable difference: the move stroke has no softness. The stroke segment that is being interacted with is always displaced with the full amount of the brush movement. As the stroke interval is displaced, it will start to take on the form of the move stroke, as shown in Figure 20.



(a) (b)

Figure 20: Move strokes. (a) shows the move stroke in red. The blue arrow depicts the direction the stroke is moved in. (b) shows the effect of pushing the move stroke against the black line.

The move stroke can be seen as a sort of stamp tool. The stroke can define a pattern which the artist can then repeatedly apply to another line. This can take away some of the tediousness of certain actions, and create consistent, repeatable results. It can also be used as a more flexible version of the move brush since the artist can define much bigger brushes. However, there is more overhead as artists have to create the brush themselves. Still, this novel type of interaction can lead to interesting experimentation with line art.

Erasing

Erasing remains faithful to the traditional approaches. Simply dragging the eraser tip of the pen over a stroke will remove that part of the stroke. We saw no reason to change stroke erasing, apart from adding

an extra option in the form of a tap interaction. The traditional method has endured for decades and is very effective.

3.5 SCALABILITY OF LINE DRAWINGS

Artists often start with small thumbnail sketches before moving to the true sketch. In most drawing systems these thumbnails go to waste as they are too small to scale up without a major loss in quality. To solve this problem, as well as the issue of canvas resolutions, we aim to make the strokes scalable without a loss in visual quality. Lines should be sharp whether the drawing has been magnified five times, or zoomed out to the size of a post stamp.

While this kind of line quality is inherent in vector drawing systems, such systems are limited to relatively simple shapes such as circles and Beziér curves. Vector systems are not designed for freehand, pressure-sensitive input. We suggest an approach that represents strokes as triangle geometry. In order to ensure that the quality of a stroke is preserved when magnified, the refinement of the stroke geometry must be varied according to the magnification level. This also ensures that at a lower magnification, no redundant geometry is rendered.

3.6 SUMMARY

In this chapter a number of methods for stroke interaction and concepts for interface design were introduced. We discussed the design philosophy to which the system would adhere. This meant giving the interface appearance due attention while keeping interface itself clean and simple. Another important requirement was to have as few mode changes as possible. The artist should not be interrupted by the GUI.

We proposed ways in which the digital canvas can become a useful tool to the artist. The first of these is the manipulation of the canvas orientation, position and magnification through gestures. Being able to orient the canvas in a comfortable position can benefit the artist in a number of ways. Important here is the ease with which gestures allow such interactions. We also mention the integration of the load and save functions with the canvas. This provides an easy way to switch sketches without getting caught up in the interface.

Two important interface elements were introduced next. First, the menu. A circular design which resembles an abstract painters pallet was used. The menu was kept simple with two modes: draw and edit. These modes determine what kind of canvas interaction the pen performs. The menu also contains two features that manage the canvas: clear and view reset. The second element is the tap and tap marker. The tap gesture can be used to either select a stroke or create a tap marker. The tap marker fades away within a short time after it has been created. Dragging from the marker can create a multi-select or erase-line, depending on whether the pen or eraser end was used.

The interaction with strokes forms one of the main contributions of this thesis. The move brush concept gives artists the ability to alter the shape of strokes in a natural, familiar way. The brush has several

uses. It can correct existing strokes and as such avoids the need to erase the stroke. It can also be used as a drawing tool. A similar approach to stroke editing is proposed in the form of draw strokes. The artist can define any stroke as a draw stroke. The draw stroke can then be used like the draw brush, thus giving the artist a way to define custom move shapes. The most obvious function for draw strokes is to repeat a certain pattern along a stroke. In this sense it works much like a stamp.

Next we show a new approach to the classical selection border. While having the same functionality, we propose to use the gestures discussed earlier to manipulate the border. This gives the artist easy, intuitive access to all the border interactions without any mode changes. We end the chapter with a brief section on the scalability of strokes. We argue that scalability without loss of quality can have a positive contribution to the artistic process.

4 THE PROOF OF CONCEPT

The previous chapter supplied the general description of functionality present in the system. In this chapter we discuss the interesting parts of the implementation. Algorithms, data structures and technology used in the proof of concept will be explained to make the proposed techniques more concrete.

4.1 TECHNOLOGY OF THE SYSTEM

The goal was to make a proof-of-concept which included all the discussed functionality. To this end, it was important that no time would be wasted on the low-level aspects of building an application, such as getting data from user input devices. Therefore, the system was built on top of the buffer framework developed by Isenberg et al. [2006]. The framework has been written in C++ and uses the OpenGL Application Programming Interface (API), a combination which is excellent for high-speed graphics performance. The framework gives easy access to the canvas, various buffers and the implementation of GUI elements through so-called strategies. It also contains tools for mundane tasks such as texture loading. The framework itself uses the Qt GUI cross-platform application framework [QtSoftware, 1995], which handles the acquisition of user input data. This includes pen tablet data, which is non-trivial to obtain as different tablets manufacturers use different driver interfaces. Where possible, data structures and algorithms from the Standard Template Library (STL) were used. These are widely available and more robust than anything written during the project could be.

4.2 INTERACTION WITH THE CANVAS

To translate input from the artist into a valid interaction, some processing might be required. This is obvious for interactions such as gestures, but even drawing a simple line requires some evaluation of the input data. The data received from the pen tablet proved to be more noisy than anticipated. Some measures had to be taken to ensure smooth interaction with the proof-of-concept. Often, pressing or releasing the pen from the tablet would be registered several times. Extra input verifications were added to filter out these 'ghosts'.

Frame gestures are recognized from their orientation. The artist drags out a line from a gesture-sensitive area. When the line is long enough, its direction in regards to the frame is calculated to determine what kind of gesture is being performed. The reason the line has to become a certain length is to avoid misclassification of the gesture. A minimum line length $l_{min} = 5.0$ was found to work well, with l_{min} measured in screen pixels. Frame gestures and selection gestures are handled in the same manner.

Because translation and rotation are both performed on the same area, thresholds are used to distinguish those interactions. If the angle be-

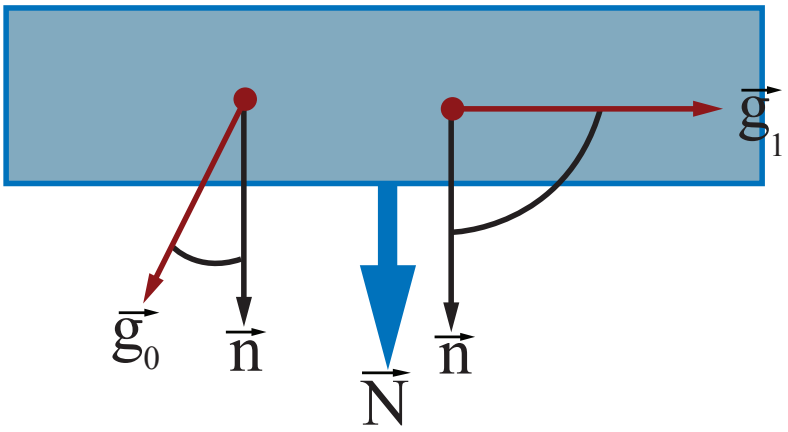


Figure 21: Canvas interaction gestures. An example of gestures performed on the 'top' frame border. Both $||\vec{g_0}|| > l_{min}$ and $||\vec{g_1}|| > l_{min}$, with $l_{min} = 5$. In the case of $\vec{g_0}$, the angle is smaller than 70° and the gesture will be classified as *translation*. With g_1, the angle is larger than 70° as the gesture runs parallel to the frame. This gesture will be classified as *rotation*.

tween the vector $\vec{g}$ of the gesture line and the edge normal $\vec{N}$ is less than 70°, the translation interaction is activated. If the angle is greater, rotation is performed instead. This means that there is a 'hard change' from one interaction to the other, with no dead-zone in between. The 70° threshold was determined through trial and error and works well in practice. For the frame gestures, $\vec{N}$ is pointing towards the center of the screen, while the $\vec{N}$ of the selection frame edges point outwards. Examples of the gesture recognition are shown in Figure 21.

4.3 STROKE DRAWING

Stroke drawing is the most basic, and vital, canvas interaction. Great care was taken in selecting the way strokes are visualized and represented internally. Efficiency is the main motivations behind the following structures and algorithms.

4.3.1 *Brush Model*

The brush model determines what the appearance of the stroke will be, based on the input from the artist. Given that the artist is using a modern graphics tablet, there are three[1] input parameters that can influence the appearance of a stroke: pressure, draw-speed and location. Location is used to determine where on the canvas the stroke is drawn, leaving us with two variables to determine the line width and opacity parameters.

1 A small portion of the commercially available tablets has a fourth variable in the form of pen-tilt. However, given the more limited availability of such tablets, this parameter is not used in the system.

The artist is given free choice in determining which variable varies which parameter. In each iteration of the render loop, the brush is sampled. While a separate thread to sample input might work better on slower computers, in practice sampling each rendered frame is sufficient. The sample is then used to create a new stroke point with the appropriate values. Determining the effect of pressure on the value of a parameter is straightforward: given pressure value $F_p \in [0, 1]$ and the maximum parameter value p_{max}, the sampled input value at a given time is

$$p_{sample} = F_p p_{max}$$

Effectively using speed is more complex. The concept used in the system is an interpretation of DynaDraw [Haeberli, 1989]. The brush is seen as a physical entity which has mass and velocity. The displacement of the brush as the artist moves it over the canvas is used to calculate the brush velocity. This is then used to determine how much of the 'paint' is deposited on the canvas. In other words, what the stroke width will be for that particular sample.

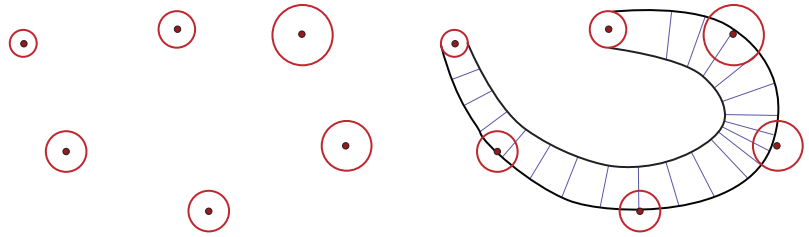
4.4 BRUSH STROKE MODEL

Perhaps the most fundamental and important part of the system is the Brush Stroke Model (BSM). The BSM determines how strokes are represented internally and how they are rendered. The BSM used in the system needs to be flexible and robust enough to be manipulated into any shape. Furthermore, rendering a stroke should be efficient: it should contain the minimal amount of geometry while maintaining an acceptable visual quality. Efficiently resampling the geometry so quality remains consistent regardless of magnification is also an important requirement.

4.4.1 *Stroke Representation*

The stroke is stored in two distinct components: its set of control points and the segments in between. The shape of a stroke is implied by the control points, which are taken as basis of a uniform cubic B-spline [Chou and Piegl, 1992]. These control points do not only store the placement coordinates, but also additional parameters such as size and opacity. The size parameter can be seen as the radius of the disc of which the control point coordinate is the center. This information is used to calculate the ribs of the stroke, which are transformed into triangle geometry. This concept can be seen in Figure 22. Several types of splines were considered to render strokes from. Most notable is the Catmull-Rom spline. Unlike the B-spline, the Catmull-Rom spline passes through the control points. While this interpolation is in theory desirable over the approximating nature of the B-spline, the difference in the number of control points needed to obtain similar visual results is not large. Also, the B-spline is C^2 (second derivative) continuous, while Catmull-Rom is only C^1 (first derivative) continuous. This makes B-spline curves flow more naturally in some circumstances. The most obvious limitation to the stroke model is that it does not allow any customization of the stroke appearance. This could be solved by extending the model with textures, or by using a procedural method to generate

the geometry along the base line of the stroke.



(a) (b)

Figure 22: Stroke representation. (a) displays the control points that define the stroke b-spline. (b) shows the (approximating) stroke that was constructed using the control points.

Encoding a stroke shape in this manner was inspired by [Seah et al., 2005; Su et al., 2002], who used a similar approach to draw digital calligraphy and Chinese ink. Given these control points, values in between can be calculated using the matrix form of the B-spline:

$$S_i(t) = \begin{bmatrix} t^3 & t^2 & t & 1 \end{bmatrix} \frac{1}{6} \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 0 & 3 & 0 \\ 1 & 4 & 1 & 0 \end{bmatrix} \begin{bmatrix} p_{i-1} \\ p_i \\ p_{i+1} \\ p_{i+2} \end{bmatrix}, t \in [0,1]$$

where $S_i(t)$ is the point p_i where the spline passes through at time t. The matrix can be worked out into the following equation:

$$S_i(t) = \frac{1}{6}(c_0 t^3 + c_1 t^2 + c_2 t + c_3)$$

with

$$\begin{aligned} c_0 &= -p_{i-1} + 3p_i - 3p_{i+1} + p_{i+2} \\ c_1 &= 3p_{i-1} - 6p_i + 3p_{i+1} \\ c_2 &= -3p_{i-1} + 3p_{i+1} \\ c_3 &= p_{i-1} + 4p_i + p_{i+1} \end{aligned}$$

Using Horner's rule [Dorn, 1961] we can rewrite the formula for $S_i(t)$ to be computed more efficiently:

$$S_i(t) = \frac{1}{6}(((c_0 t + c_1)t + c_2)t + c_3)$$

which saves six multiplications per calculated spline point, as the calculation has to be done for both x and y coordinate components.

4.4.2 *Refining Stroke Geometry*

Stroke refinement helps preserving the visual quality of a sketch when it is enlarged. As such, it is a very important aspect in creating scalable artwork. Figure 23 shows what kind of impact stroke refinement can have. While vector-based system have, by nature, scalable artwork, they lack accuracy or an intuitive interface when it comes to freehand sketching. The stroke model in our system solves both problems by combining accurate stroke representations and an established drawing interface.

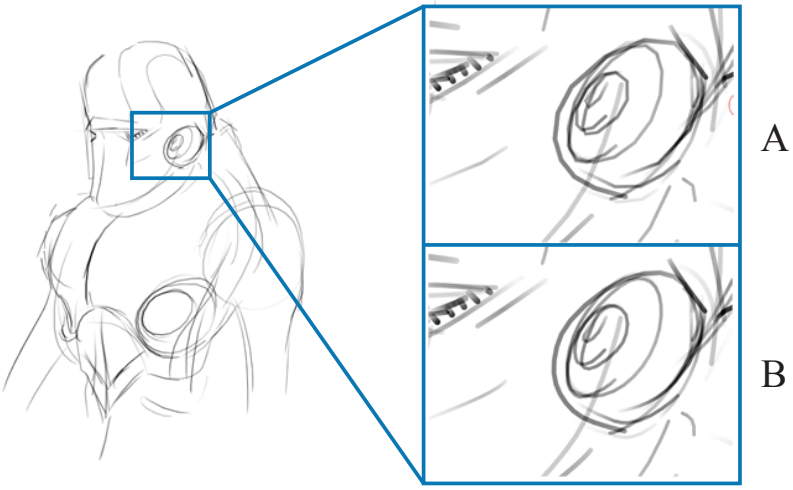
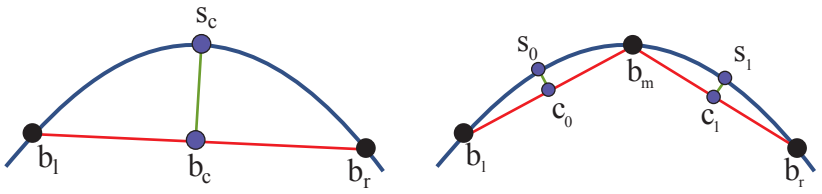


Figure 23: Stroke refinement. A is an magnified version of the sketch. B is a magnified version with stroke refinement.

The refinement of the stroke geometry can be efficiently recomputed from the stroke segments. Segments store ribs of the stroke at a certain interval and are pre-computed from the control points using an adaptive subdivision scheme. This is a popular method to compute splines [Rappoport, 1991]. The initial borders of the subdivision are two neighbouring control points. At each step of the subdivision, the distance between the center of the straight line defined by the borders of the subdivision region and the center of the b-spline segment through this region are compared. If the distance is greater than some constant σ, the difference between the subdivision and the true B-spline is still too great and the subdivision continues by subdividing at the middle. This process is visualized in Figure 24. A smaller value for σ leads to a closer approximation, but also more ribs and, thus, more geometry.

The choice of σ depends on the scale at which the stroke must have an acceptable visual quality. The formula $\sigma = \frac{\varepsilon}{z}$ is used, where z is the scale and ε is some constant. Finding a suitable value for ε is mostly an exercise in trial-and-error. A value of $\varepsilon = 0.7$ was found to work well.

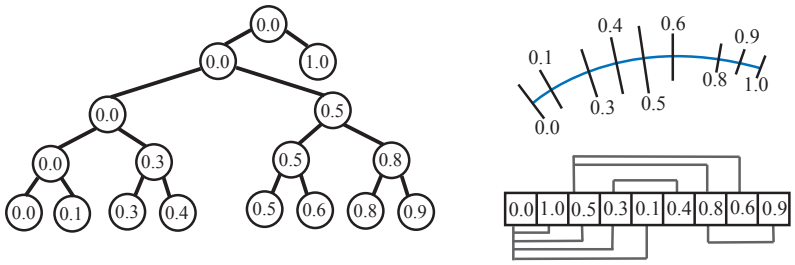
A stroke segment is comprised of a merge tree that stores the ribs in its nodes. Merge trees have been successfully used in Level of Detail (LOD) methods. In particular, El-Sana et al. [1999] use merge trees to expand and collapse vertices of triangle strips. A merge tree has the left-most rib $r_{0.0}$, which corresponds to $t = 0$, as its root. The root is then given the



(a) (b)

Figure 24: Adaptive subdivision. (a) shows the initial situation with boundaries b_l and b_r. The length of $b_c s_c$ is compared against σ to see if recursion should continue. In (b) the recursion has gotten one step further. The original bounds have been split at b_m.

rib $r_{1.0}$ as its first child. The ribs produced by the subdivision are added to the tree one-by-one. A suitable place in the tree is found through recursion: a rib r_t is compared against the children of the current node n_c, which initially is the root. Two situations can occur: if the t value of r_t is greater than that of the child it is being compared against (r_t is to the right of the child), this child becomes the new current node m_c. Now, r_t is compared against the children of m_c, repeating the process. If m_c has no children, r_t becomes the first child of m_c. In the second situation, r_t was smaller than the current child. Now r_t is compared against the next child of n_c, and the process is repeated. If there are no more children left, r_t becomes the latest child of n_c. An example of a possible mergetree and the corresponding line segment is shown in Figure 25.



(a) (b)

Figure 25: Merge trees for stroke representation. (a) shows an example of a possible merge tree, while (b) shows the corresponding line segment. (b) also shows a different visualization of the parent-child relation between the nodes. While (a) might give a different impression, nodes are not actually stored more than once.

To retrieve an ordered set of ribs from the merge tree it must be traversed in a depth-first manner, as shown in Figure 26. To vary the set of ribs used to display the segment, an extra variable is added to each node: the error ξ of the spline approximation corresponding to the rib stored in the node. ξ is used in the subdivision step and is the distance from the approximation to the real curve. During the depth-first traversal ξ is compared against the current scaling factor (or zoom level) of the canvas. If ξ is not too small for the current zoom level, that is to say the rib has a noticeable impact on the visual quality, it is added to the

displayed geometry. Because the node insertion order is the same as the subdivision order, ribs with a smaller error are propagated further down the tree. This means that if the ξ of a child will be smaller than that of its parent.

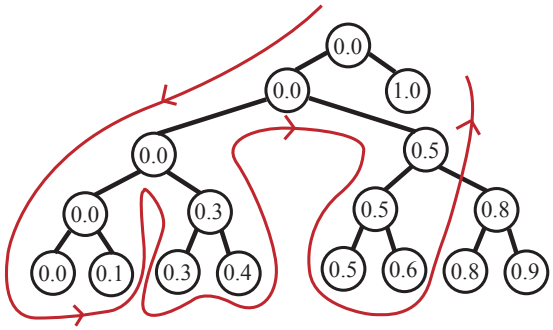


Figure 26: Merge tree traversal. A possible depth-first traversal of the merge tree. In this case the error value of nodes 0.8 and 0.9 are too small.

4.4.3 *Sampling Strokes*

There are a number of situations where it is desirable to reduce or increase the number of control points in a stroke. These downsampling and upsampling methods are an important step in many of the systems functions, such as rendering and editing strokes.

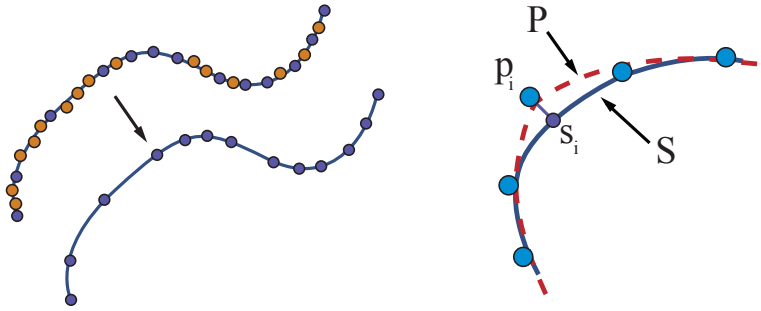
Downsampling

User input is sampled each frame. This produces an over abundance of information. For instance, when the artist draws a stroke many of the sampled stroke points have a contribution to the stroke which has no visual impact. To efficiently process and render strokes it is important that these points are removed from the stroke. To that end we perform a downsampling operation.

Given that a stroke is simply a sequence of control points, strokes can be resampled in the same manner one would resample a set of points. In particular, the approach chosen for the system was based on a sampling technique used in point set surfaces discussed by Alexa et al. [2008]. In short: given a set of points P, we compare the curve implicitly formed by the sequence of points in P against the curve of $P - p_i$ with $p_i \in P$. If the difference is small, p_i can be discarded. A simple example of downsampling is shown in Figure 27.

Completely comparing both lines against each other is is inefficient and not necessary. Each p_i is compared against the local neighbourhood without p_i. A local neighbourhood can be used because a B-Spline is only influenced by local control points. The spline S formed by p_{i-2} p_{i-1}, p_{i+1} and p_{i+2} is computed and the minimal distance between S and p_i is calculated. This distance is the approximate difference between P and $P - p_i$.

If the distance is smaller than some threshold value ξ, p_i is discarded. Choosing a large ξ leads to an interesting side-effect: it smooths the



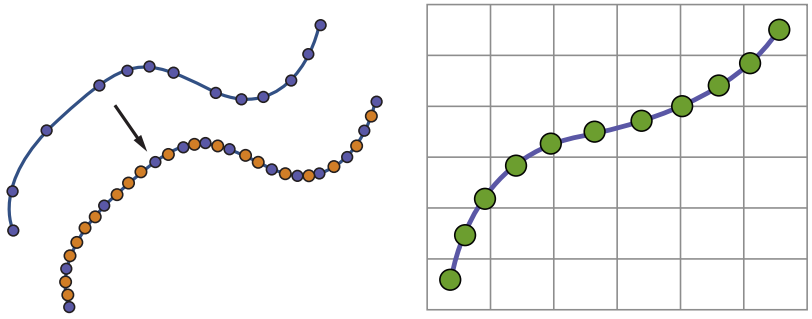
(a) (b)

Figure 27: Downsampling strokes. In (a) the yellow points have a low impact on the spline shape and are discarded. (b) shows the local neighbourhood around p_i. The dashed red line P is the original stroke shape with p_i included. S is what the shape will look like without p_i. The length of $p_i s_i$ is compared against ξ.

user input. This could be useful for artists who do not have a steady hand, yet want to draw long, smooth curves. For that reason the value of ξ can be defined by the user through the menu, where it is referred to as the stroke smoothness value.

Upsampling

Normally, the set of control points of a stroke will be as sparse as possible. However, for some stroke interactions such as displacing a stroke with a move brush or zooming in beyond the already available amount of detail, the set of points must be upsampled to approximately 1 point per screen pixel to get an accurate result, as shown in Figure 28b.



(a) (b)

Figure 28: Upsampling strokes. In (a) the yellow points have been generated to get a denser point approximation to the spline curve. (b) shows a grid which represents a small portion of screen pixels. Approximately one spline point per pixel is generated.

In the case of splines, upsampling is a relatively easy procedure: we compute more points on the spline, using the existing control points as input for the spline equation. While the matrix form of the spline formula could be used, it is not the most efficient method in this case. The spline points to be computed are equally and closely spaced. In such a

situation the forward differencing technique [Bruijns, 1998] can be used.

Forward differencing is an efficient way for evaluating a polynomial at a high number of uniform steps. For example, to evaluate a cubic polynomial $p(t)$ at uniform steps, one derives a difference function that expresses the difference between $p(t)$ and $p(t+\tau)$ where τ is the step size. In the case of a cubic polynomial, the difference is a quadratic polynomial. More formally, the forward difference is a finite difference defined by

$$\Delta\, a_n \equiv a_{n+1} - a_n$$

This finite difference operator can be applied repeatedly to obtain higher order finite differences

$$\Delta^2\, a_n = \Delta^{k-1}\, a_{n-1} - \Delta^{k-1}\, a_n$$

For the second forward difference this works out to

$$\Delta^2\, a_n = \Delta_n^2 = \Delta(\Delta_n) = \Delta(a_{n+1} - a_n) = \Delta_{n+1} - \Delta_n = a_{n+2} - 2a_{n+1} + a_n$$

A quadratic polynomial can be evaluated more efficiently than a cubic polynomial. However, this technique can be applied to a quadratic function as well. The result is a linear function. Applying the technique one last time on the linear function produces a constant. Given these forward differences, a polynomial can be evaluated by repeatedly adding up the differences. Applying forward differencing to a B-spline results in the following forward differences:

$$\begin{aligned} p &= \frac{1}{6}c_3 \\ F_1 &= \frac{1}{6}(c_0\tau^3 + c_1\tau^2 + c_2\tau) \\ F_2 &= \frac{1}{6}(6c_0\tau^3 + 2c_1\tau^2) \\ F_3 &= \frac{1}{6}(6c_0\tau^3) \end{aligned}$$

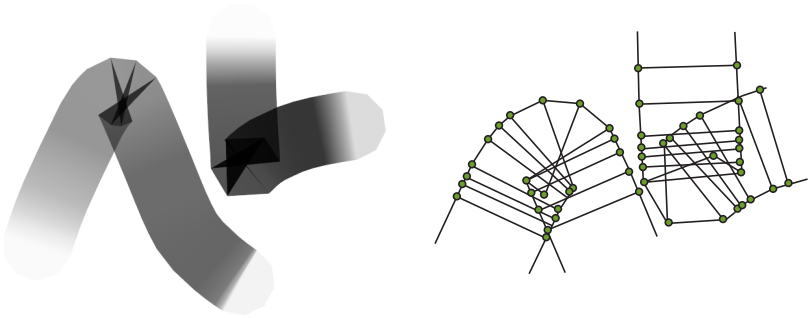
where $c_i, \forall i$ in $[0,3]$ are the B-spline coefficients and p is the current point along the spline. After initializing the differences, each new point can be generated by calculating

$$\begin{aligned} p &= p + F_1 \\ F_1 &= F_1 + F_2 \\ F_2 &= F_2 + F_3 \end{aligned}$$

each iteration of a loop.

4.4.4 *Stroke Appearance*

One of the inherent problems of a geometric stroke representation is that the stroke can fold in on itself, which leads to visual artifacts such as those shown in Figure 29. The system performs an (optional) pass over the stroke geometry that eliminates many of the folds. The pass is



(a) (b)

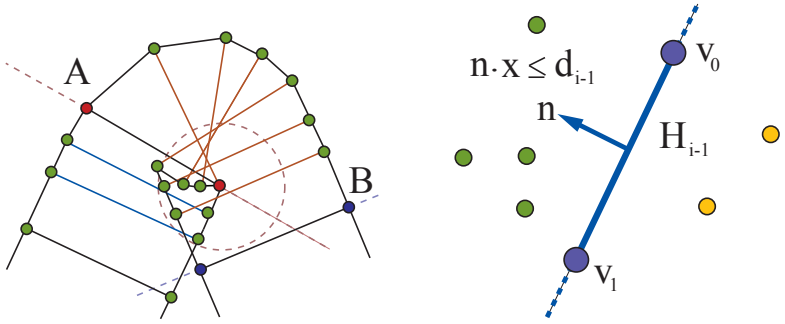
Figure 29: Shown in (a) are two examples of folding errors which can become noticeable with the use of big brushes and variable opacity. In (b) a simplified version of the underlying geometry of the folds in (a) is shown.

left optional so less powerful computers can still run the system.

The idea is to iterate over all the stroke ribs until a rib r_i is found that has an endpoint which is located 'behind' the previous rib r_{i-1}. The rib r_{i-1} can be identified as rib A, the last rib before the stroke starts to fold in on itself. To find out if endpoint v_0 or v_1 of r_i is on the wrong side of r_{i-1}, we compare v_i with the halfplane that passes through r_{i-1}:

$$H_{i-1} : \{x \in R^2 \mid \vec{n}x - d_{i-1} = 0\}$$

where x is the point being compared, $\vec{n}$ is the normal of $v_0^{i-1}v_1^{i-1}$ and $d_{i-1} = v_0^{i-1}\vec{n}$. In the case of the system, r_i is on the wrong side if $\vec{n}v_j^i - d_{i-1} < 0, j \in [0,1]$. A visualization of this can be seen in Figure 30b.



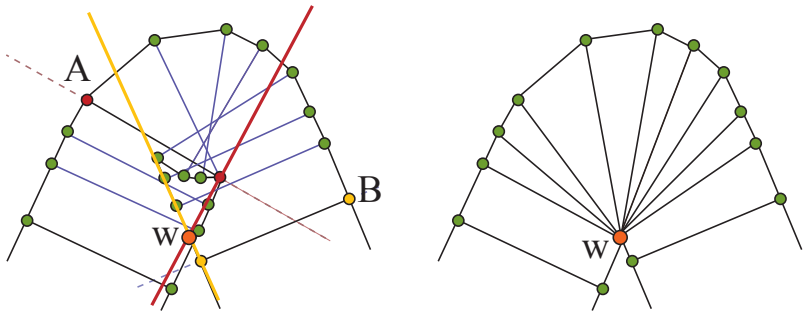
(a) (b)

Figure 30: Fold detection. (a) depicts a possible fold occurrence. Orange ribs have a vertex on the wrong side of halfplane A, while blue ribs are on the wrong side of B. If the vertices fall outside the dashed red circle, they are not considered on the wrong side. This is a safeguard for situations where large parts of a stroke overlap. (b) shows an example of halfplane H_{i-1} defined by vertices v_0 and v_1.

Once A has been found, the algorithm iterates over the ribs until a rib r_b is found that no longer has an endpoint on the wrong side of A, or is too far away from A. The rib r_b defines halfplane B. Up to this point, around half of the offending ribs will have been found. The previous

operation is now mirrored with halfplane B to find the remaining half, starting from the rib before A and working backwards. The resulting situation is shown in Figure 30a

When all the ribs have been found, the offending endpoints are repositioned to a common point w. The way in which w is calculated depends on the situation: if the angle between r_A and r_B is big, the intersection point between perpendicular lines of both ribs is calculated and used as w, as shown in Figure 31. If, however, the angle is small, the intersection point will be located far from the actual fold and give a poor solution. In such cases an endpoint of r_B is used as w, which will give a reasonable result.



(a) (b)

Figure 31: Fold repairing. In (a) the perpendicular lines of A and B which result in intersection point w can be seen. (b) shows the rebuilt geometry, where w was used as common vertex.

While suitable for common small folds, there are situations where the method behaves poorly. These situations, such as very sharp bends that cause large portions of the stroke to overlap, are best left untreated. Therefore, the method cannot get rid of all the artifacts. However, the situations where it fails are very difficult to resolve as the artist might have intended a certain amount of overlap in the stroke.

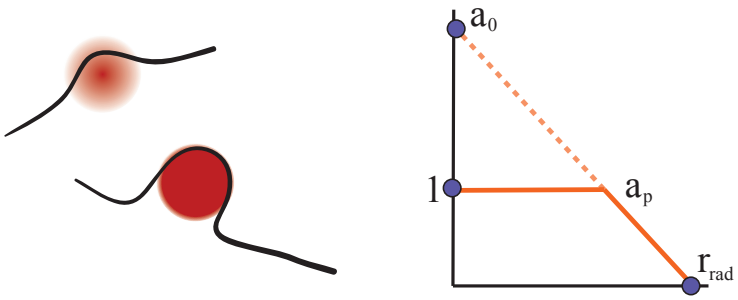
4.5 STROKE SHAPE EDITING

Stroke shape editing is a fairly complex interaction. First the strokes affected by the operation have to be found in an efficient manner, as brute force methods become slower as the number of lines in a sketch grow. Next the sparse stroke data has to be upsampled to obtain an accurate interaction result. The operation is then performed on the data and, finally, the stroke is downsampled again. We start with the interactions and how they change the stroke shape.

4.5.1 *Editing Strokes with a Move Brush*

To move strokes, or parts thereof, with a brush posed an interesting problem. A 'move brush' representation needed to be developed that is both robust and flexible. To this end, the move brush was given properties that determine the strength and reach of its area of effect.

This gives the artist control over how much a stroke is moved in regard to its distance to the brush. This can be seen in Figure 32a.



(a) (b)

Figure 32: Move brush parameters. (a) shows the effect softness has on the move brush, while (b) illustrates the radial falloff formula.

These properties can be defined with two parameters which can be changed with the menu: brush size and brush softness. The purpose of the brush size parameter is quite straightforward. Brush softness affects the way the stroke is moved as it comes in contact with the move brush. The move brush uses a *motion transference value* α to determine how much of the brush motion is applied to the stroke. Softness determines the radial falloff of α. The value of α lies in the range $[0, 1]$. Given the brush movement vector $\vec{v}_m$, move brush M and stroke S, the displacement $\vec{\omega}$ of a point $p_S \in S$ can be calculated as

$$\vec{\omega} = \vec{v}_m \alpha_{ps}$$

where α_{ps} is the transference value of the move brush at the position of p_S. Maximal softness will result in a brush that has a linear falloff for α from $1 \rightarrow 0$ as the distance from the radial center c_{rad} goes towards the brush radius r_{rad}. Minimal softness results in a brush with no falloff and $\alpha = 1$ for the whole brush area.

Calculating α_p can be done with the following formula

$$\alpha_p = \alpha_0 - (\alpha_0 \frac{d}{r_{rad}}),\ with\ \alpha_0 = \frac{r_{rad}}{softness},\ d = ||c_{rad} - p_S||$$

The value of α_p is clamped to the $[0, 1]$ interval. Figure 32b shows an example of how this formula behaves. While the brush can contain multiple radials, resulting in a variety of move shapes, the brush in the implementation only contains one radial to create a close analogue to the drawing brush.

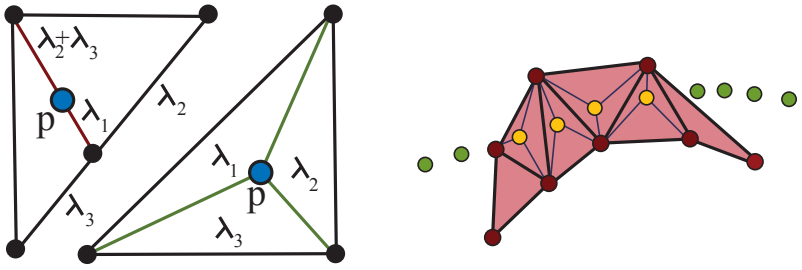
4.5.2 *Editing Strokes with a Move Stroke*

The move stroke, while very similar in function to the move brush, has a different implementation. Approximating a move stroke with the radials used with move brushes would create an inaccurate result, as radials cannot approximate a locally smooth surface well. A more general approach was needed: to calculate the displacement, the triangle strip geometry of the original stroke is used. The principle to move points is inspired by the move brush. Given a stroke S, the point $p_S \in S$

is moved with the full displacement $\vec{v}_m$ of the move stroke M if p_S is inside any triangle $abc_i \in M$. Barycentric coordinates [Bottema, 1982] are used to determine if a point lies inside a triangle: given triangle abc, the barycentric coordinates of point p with regard to abc can be calculated as

$$\begin{aligned}
\lambda_0 &= (b_x - a_x)(c_y - a_y) - (c_x - a_x)(b_y - a_y) \\
\lambda_1 &= \frac{(b_x - p_x)(c_y - p_y) - (c_x - p_x)(b_y - p_y)}{\lambda_0} \\
\lambda_2 &= \frac{(c_x - p_x)(a_y - p_y) - (a_x - p_x)(c_y - p_y)}{\lambda_0} \\
\lambda_3 &= \frac{(a_x - p_x)(b_y - p_y) - (b_x - p_x)(a_y - p_y)}{\lambda_0}
\end{aligned}$$

Notable properties of barycentric coordinates are that $p = \lambda_1 a + \lambda_2 b + \lambda_3 c$ and $\lambda_1 + \lambda_2 + \lambda_3 = 1$. If $0 \leq \lambda_i \leq 1$, $\forall\, i$ in $[1, 2, 3]$, point p is inside the triangle, or on one of the triangle edges. If we find one triangle in the triangle strip for which this condition holds, p is considered inside M.



(a) (b)

Figure 33: (a) Barycentric coordinates. Shown are two ways in which barycentric coordinates can be interpreted. (b) Move stroke geometry: yellow points are classified as inside the stroke (and should be moved), while green points are outside the stroke.

The move stroke comes with one noticeable downside. Because collision between a stroke interval and the brush depends on the overlap between the volume of the move stroke and the stroke interval, a thin move stroke might produce unpredictable results. When the brush is moved quickly, collisions might not be detected as parts of the brush have moved through the stroke interval completely without overlapping it at some point. A possible way to solve this would be to use a more robust collision detection scheme, such as those used with rigid body physics [Hadap et al., 2004].

4.6 SPATIAL PARTITIONING OF STROKES

Many of the functions in the system try to find the stroke closest to a specific point or strokes within a certain spacial range. A naïve, brute force approach would be to compare against all the strokes on the

canvas. For a sketch of reasonable complexity this would be too slow. A family of algorithms designed to speed up range finding or nearest neighbour queries are the space-partitioning algorithms.

4.6.1 *Choosing a Spatial Partitioning*

There exist a great variety in space-partitioning, ranging from simple grids to more complex algorithms such as a Binary Space Partioning (BSP). Two well-known and relatively simple partitioning schemes were examined: quadtrees and KD-trees. Each partitioning has situations where it performs well and situations where it performs poorly, so care had to be taken to choose a scheme that is best suited for line drawing data. Creating an optimal KD-tree can be very complex, and may lead to the algorithm not being very suitable for dynamic data [Wald et al., 2006], while quadtrees perform badly when the data is very evenly distributed [Jain and Shneiderman, 1994]. However, evenly distributed data is unlikely to be the case in a sketch. Therefore, a simple quadtree was chosen.

4.6.2 *Application of a Quadtree for Line Data*

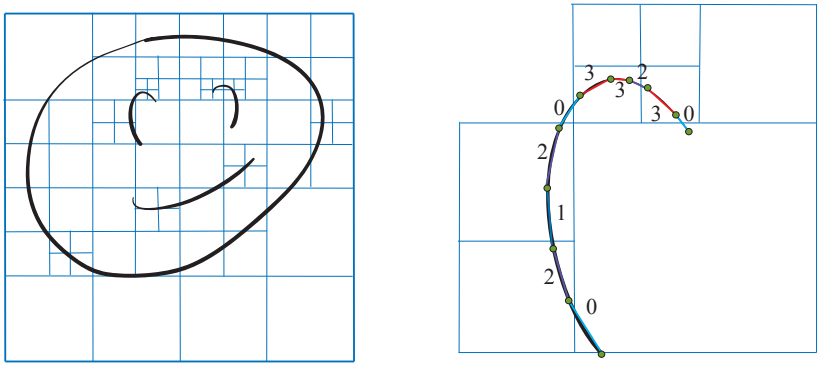
First, we establish what a quadtree is: a quadtree is a 2D version of the more general octree. Each node, or square, in a quadtree can be subdivided into 4 smaller squares. Each of these children covers $\frac{1}{4}$ of the area covered by its parent. Regions are subdivided recursively when the need arises, such as when the bucket of a square is full. While often called squares, regions may be rectangular, or any shape for that matter.

A regular quadtree usually stores points. This would naturally lead to the idea of storing the control points of a stroke inside the tree. However, this is not sufficient. The space between control points can be arbitrarily large; a stroke could intersect a query range without having a control point within the range. A way to solve this is to store linear line segments instead. These segments approximate the shape of the stroke between control points. A segment is stored as two end-points and a pointer to the stroke it approximates. Figure 34 shows an example of this.

To store line segments we start at the root of the tree. Recursively, the child is chosen that covers the line completely, until the smallest such square has been found. The segment is stored in that square. A special case arises when storing lines in a quadtree: the smallest square might not be a leaf node. That is to say, a line segment might be stored in a node somewhere in the middle instead of at the bottom level of the tree. Such a segment intersects two squares. In practice such segments seldom appear at early levels of the quadtree because the strokes are approximated by a large set of segments.

4.6.3 *Range Queries*

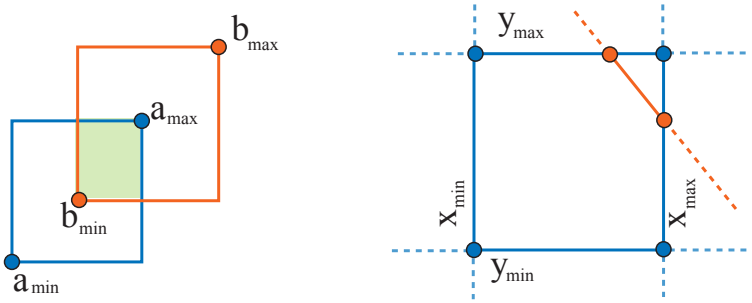
Range queries are used to find strokes that intersect a certain area. Such a range is defined by an axis-aligned bounding box. Starting at the



(a) (b)

Figure 34: Quadtree details. In (a) the strokes of a simple drawing have been added to a quadtree. (b) A closeup of part of the quadtree with the segments that approximate the black stroke. Segments that are more red have been stored deeper in the tree, while blue segments are stored at earlier levels, because they overlap several squares. The numbers also indicate depth of the corresponding segments.

root, the range is compared against each of the four children. If a child (partially) overlaps the range, its children are tested against the range as well. This recursion continues until there are no new children to compare against the range. The line segments in the buckets of children that overlapped the range are added to the list of results. Because partial overlaps are also included, all resulting segments are compared against the bounding box to confirm that they are inside the range.



(a) (b)

Figure 35: Intersections. (a) depicts a typical overlapping bounding boxes situation between boxes a and b. In (b) the box-line overlap test can be seen: the line is clipped to each of the four slabs. In this case there is still a line left after the procedure, so it overlaps the bounding box.

Checking if the bounds b_R of range R overlaps the bounds b_S of square S is quite straightforward: because the bounds are axis-aligned the minimal and maximal coordinates of the bounds can be compared, so if the condition

$$(b_R^{max} \geq b_S^{min}) \wedge (b_R^{min} \leq b_S^{max})$$

holds, the boxes overlap. An example of this is shown in Figure 35a Checking whether a segment overlaps a box is a more complicated

process. The line is clipped along each of the infinite lines that run parallel to the edges of the box, often called slabs. If there is still a line segment left after it has been clipped, the line overlaps the box. This process can be seen in Figure 35b.

4.7 SUMMARY

In this chapter the various algorithms and data structures used in the concepts discussed in Chapter 3 were presented. Starting with a description of the BSM, it was shown how this model is used to create scalable strokes. Here, an emphasis was put on efficiency. We continued on by presenting methods for upsampling and downsampling strokes, an operation which is used with many of the stroke interactions. Before moving on the stroke interactions, a novel method to remove folds from the stroke geometry was given.

Next the stroke interactions were discussed. The move brush and the principles regarding point displacement were presented. We showed how the amount of displacement can be varied through user-definable brush parameters. Finally, the implementation of the move stroke was given, and it was shown how it differs from the move brush because it uses stroke geometry to define a brush.

We ended the chapter with a discussion of the spatial partitioning used to quickly find strokes. The choice of the quadtree was reviewed and a brief overview was given on storing line segments and querying the tree.

5 DISCUSSION: CREATING SKETCHES

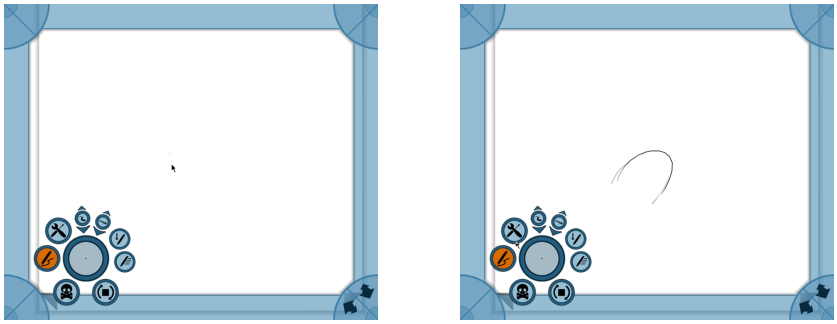
The discussion forms an important part of this thesis, as the merit of the presented concepts and methods can only be determined by the people who use them. We examine the practical benefits of the implementation in several ways. First, we perform a case study. This study will illustrate the use of the developed tools in a real life situation. Second, we analyze an informal user study which was performed. The implementation was evaluated with two groups; artists and people with no artistic background. Third, we analyze how the implementation was used to create certain sketches. Finally, we show some of the works that have been created with the implementation.

5.1 CASE STUDY

In order to illustrate the concepts presented in this thesis, we examine, step by step, how a simple sketch was made using the implementation and its features.

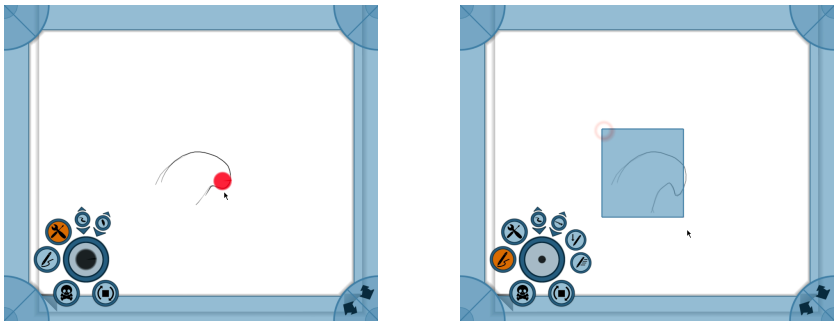
5.1.1 *Sketching a Creature*

The sketch was recorded with movie capture software. The images here represent a time lapse of that movie, highlighting the methods used to accomplish important steps in the sketch. For illustrative purposes, the sketch was kept simple. The process can be seen in Figure 36 through Figure 42.



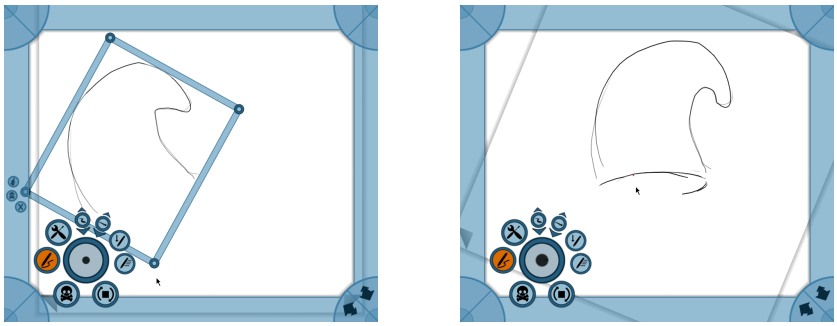
(a) (b)

Figure 36: **(a)** The starting point. This is the view presented to the artist when the implementation is started. **(b)** The first lines. The artist uses a static brush size and varies opacity with pressure, the settings he or she is most comfortable with. The artist starts small, because the shape is easier to draw that way.



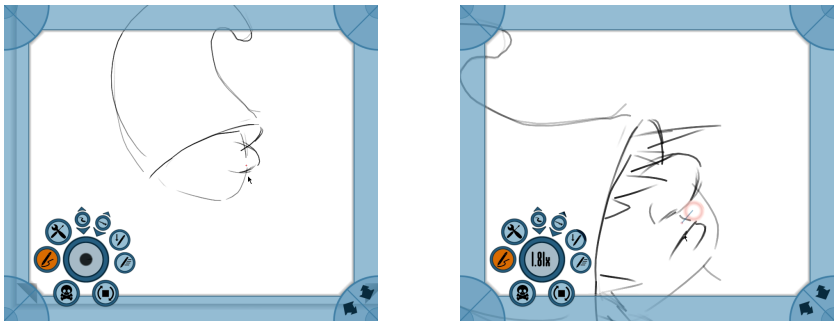
(a) (b)

Figure 37: **(a)** The shape does not resemble the artist's intent. In this instance he uses the move brush to change the shape he is currently drawing. The tool is used as both a correctional technique and drawing technique simultaneously. **(b)** The shape is finished, but very small and at the wrong place. To apply a transformation to the lines, they are selected with the tap-based multi-select.



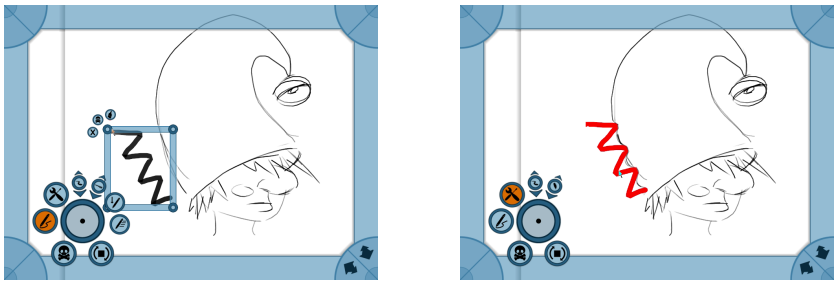
(a) (b)

Figure 38: **(a)** The result of the transformation. The shape has been translated and scaled, without losing any line quality. **(b)** The artist rotates the canvas with the frame gestures so he can more comfortably draw the lines. The drawing motion he is comfortable with now lines up better with the line he wants to sketch.



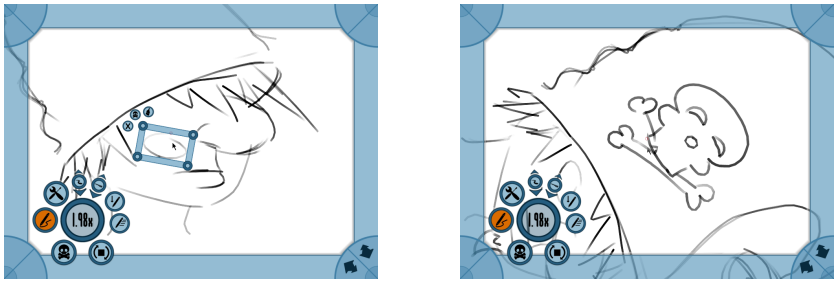
(a) (b)

Figure 39: **(a)** Some more lines have been drawn to flesh out the character. After drawing the nose, the artist uses the eraser side of the stylus to to erase parts of the line that intersect the nose of the character. **(b)** Again, the canvas is rotated so certain lines can be drawn more easily. This time a complete stroke is removed with the line-erase tap interaction.



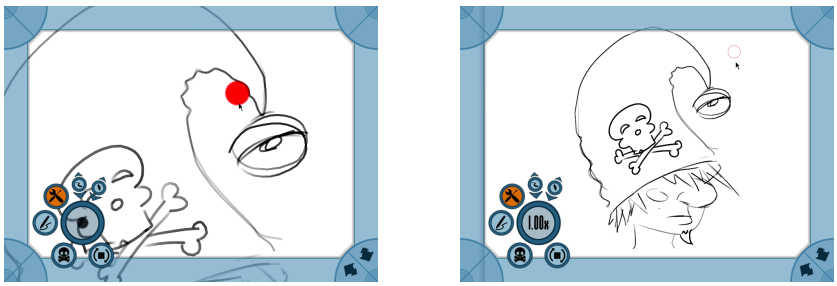
(a) (b)

Figure 40: **(a)** To use a move stroke, the artist first draws the shape he wants to move things with. The stroke is then selected and the 'make move stroke' button along the selection border is pressed. **(b)** By using the eraser tip of the styles while in edit mode, the artist uses the move stroke to redefine strokes with a more complex shape.



(a) (b)

Figure 41: **(a)** Here a stroke has been tapped to select it. The selection border is then used to translate the stroke. **(b)** The canvas has been rotated again to ease the drawing of certain lines.



(a) (b)

Figure 42: **(a)** Some more corrections to the hat are done using the move brush. **(b)** An overview of the final image.

5.2 INFORMAL USER STUDY

An informal user study was conducted to receive feedback regarding the concepts and techniques discussed in this thesis.

5.2.1 *Participants*

The study was conducted with two distinct groups of participants. The first group consisted of artists, while the second group contained people with little to no drawing experience. A wide range of backgrounds was found in both groups. The 'artist group' included an experienced oil painter with no prior digital drawing experience, an artist who draws digitally almost exclusively, a professional industrial designer, a professional animator and two art students. The 'non-artist' group contained a human-machine interaction specialist and several people with some prior drawing experience, but who have never drawn digitally. Ages within the the artist group varied between 22 and 55 (M : 32.5, Mdn : 29.5 and SD : 12.06), while the ages of the non-artist group were between 24 and 53 (M : 31.5, Mdn : 24.5 and SD : 14.34).

5.2.2 *Setup*

The evaluations were conducted in the following manner: the participant was placed behind a laptop with an A6 Wacom tablet, which had the implementation running (Figure 43). The participant is then told to experiment with the program, while voicing his or her thoughts. During this phase, the participant has had no explanation regarding the interface. After being sufficiently observed during this phase, the interactions and tools are explained. The participant is encouraged to use the tools to create a sketch, and is often engaged in a dialog with the evaluator regarding the functionality and possible improvements. Afterwards, the participant is asked to fill in an evaluation form which rates each functionality. During the whole process, the evaluator is making notes regarding the behavior and suggestions of the participant.

5.2.3 *User Observation and Impressions*

While the basic drawing mechanics were quickly discovered by the participants, the advanced tools and the use of gestures needed to be explained, as there is no indication to such operations. After receiving the explanation, participants started performing the gestures, and quickly gained adequate skill in using them. For some, there was an initial confusion between using the frame borders to translate or rotate, but this resolved itself quickly. Without exception, the participants saw the frame gestures as a useful interaction which they missed in other systems. Especially the rotation of the canvas was liked. Participants were quick to adapt the gestures into their work flow.

A common theme during the evaluations was the participants' attempt to use interactions and techniques from other drawing system to interact with the implementation. This was especially true for the artist group, as they have spent a great deal of time using and mastering these systems. With a lack of additional info, they tried to map their

existing knowledge to the new situation. A striking example of this was participants trying to undo strokes with the standard undo hot-key combination. This knowledge extrapolation indicates that, for users to quickly learn a new system, a great degree of familiarity and conformity might help. These 'tried and true' models could be used to ease the user into the new concepts.

The implementation proved to be more suitable for some sketching styles than others. Participants who sketched using a fast succession of short strokes sometimes inadvertently created a tap marker. While participants who had this problem were able to adapt, it did affect their work flow. Apart from this issue, all participants were able to create sketches in their own style. They were not limited by the implementation in that regard.

(a) (b)

Figure 43: Participants who are exploring the interface.

The vectorized approach to strokes was generally well received. Some participants found that the vectorization sometimes resulted in a different line than they intended. However, this was accepted as a minor inconvenience. The quality of the scalable strokes was found to be good by most participants, but some missed the option to give a stroke a texture.

The tap marker was the most difficult interaction for the participants to become proficient in. Partially due to the way in which a pen tablet functions [Apitz and Guimbretière, 2005], sometimes an interaction with the tablet surface which was intended as a tap would result in a drag instead. No tap marker would be created in such cases, which confused participants. Some participants were better at performing the tablet tap than others. The amount of experience with a tablet did not seem to be a factor, as the problem arose with artist and non-artist participants alike. Participants did seem to improve at this interaction as they used it more.

Reactions regarding the page-flip saving method were very diverse. While some regarded it as an interesting solution, others disliked having to move the canvas in order to save. One participant found that this functionality was too close to the drawing area and might cause him to change sketches accidentally. Given such extremely varying responses, it is difficult to give a final verdict on this concept. Regarding the menu

system, some participants had trouble understanding the meaning of certain icons. Overall, the icons were found to be recognizable once their purpose was made clear.

Interesting was the difference between both participant groups. While the non-artist group approached the implementation more as a toy, the artist group saw it more as a possible replacement for a system in their current work-flow. This resulted in a different mindset when evaluating the concepts. Non-artist participants were more interested in the 'fun' factor of the interactions, where intuitiveness was an important quality. The artists were more concerned about the effectiveness of the tools. They also looked more for conformity, as they compared the interactions to those in systems which they were already using.

5.2.4 *Suggested Improvements*

Many of the participants gave useful suggestions during the evaluation process. Two suggestions were very common; an undo button and menu tooltips. The absence of these was felt by most participants. The use of tooltips has often been proven and their inclusion would make the interface a great deal easier to navigate without any prior knowledge. The undo key was found to be an important part in the work-flow of some participants, and could not be replaced by the novel stroke editing options.

While one of the ideas of this thesis is to use only the pen tablet for interaction, participants from the artist group did miss the inclusion of hotkeys. They felt hotkeys were integral to their workflow and an ideal way to use their offhand. A few of the artists also brought up the matter of layers. They argued that having multiple drawing layers would make it easier to use construction lines in the sketching process. One artists mentioned a 'wand' selection tool in order to make more detailed selections. Also, instead of moving the menu with the tap marker, some participants suggested being able to drag the menu using the center circle.

5.3 IMPLEMENTATION EVALUATION

While observations are an important tool in understanding the results, it is difficult to measure them. Therefore, we will now discuss more quantifiable results obtained from the evaluation process.

5.3.1 *Functionality Evaluation*

Aside from the interaction with the evaluators, participants of the informal user evaluation were also asked to fill in an evaluation form. On the form, participants could rate each of the features present in the implementation according to their overall opinion. The results can be seen in Table 1.

What is immediately apparent from the evaluation is that the results from the non-artist group and the artist group are very close to each other. The artist group graded slightly lower in certain areas, but the

How much do you like...	– –	–	–/+	+	++	△	□
Overall Appearance			△		□	3.4	4.6
Rotating Canvas using the Frame					◯	5	4.6
Moving Canvas using the Frame				△	□	4.4	4.6
Zooming using the Frame Corners				△	□	4.2	4.6
Saving by using Canvas Page Corners			△	□		3	3.8
Using the Move Brush				◯		4.4	3.8
Creating a Move Stroke from a Selection			△	□		3.4	3.6
Using a Move Stroke			◯			3.2	3.4
Transforming a Selection				◯		4	3.8
Selecting Lines through Tapping			△	□		3.4	4
Using the Tap Marker to Select				◯		3.8	4.4
Using the Tap Marker to Erase				◯		3.8	4.4
Move the Menu with the Tap Marker	△	□				2.2	3.4
Using the Menu				◯		3.8	3.8
Resetting the Canvas View					◯	4.6	4.6

Table 1: Results of the informal user evaluation: strongly dislike (– –), dislike (–), neutral (+/–), like (+) and really like (+ +). The artist group is denoted with a △ and the non-artist group with a □ . If both groups have the same result this is indicated with a ◯ . Results are the average of the evaluation forms turned in by the participants. The last two columns show the exact grade of the artists and non-artists, respectively.

only large disparity is found in the "*Overall Appearance*" mark. This is at least partially because many participants from the artist group found the blue screen border too large.

As mentioned before, the frame gestures themselves were very popular with all the participants. This is likely the reason why the ability to reset the canvas view scored high as well; it complements the gestures. Other concepts that were liked by both groups are move brush tool, multi-select and erase-line. The final tap marker interaction, moving the menu, was one of the least liked features. This is likely the result of the interaction sometimes happening when the participant intended to do something else.

Another feature which was not well-received is the move stroke. The biggest issue was the effort it takes to create the move stroke. Some participants would rather use the move brush as it can perform similar operations, but can be accessed more easily. Other participants found the move stroke slightly cumbersome in usage.

Overall, many of the concepts were liked by the participants. Many of the functionalities which received a low rating could be substantially improved with the feedback from the participants and further design iterations, in particular the tap marker and the menu. For some of the concepts, such as the move stroke and the canvas page corners, it is questionable whether further research would be a wise investment.

5.3.2 *Workflow Analysis*

To analyze how the different interactions were used during the sketching process, the implementation records each execution of an interaction as they are performed by the artist. Examples of command execution plots can be seen in Figure 44 and Figure 45. While the plots are from different artists, with different styles, they are remarkably similar. From these plots we learn that certain interactions, namely drawing and manipulating the canvas, are used throughout the sketching process. The fact that canvas interactions are used so frequently indicate that they integrate well with the workflow of the artist.

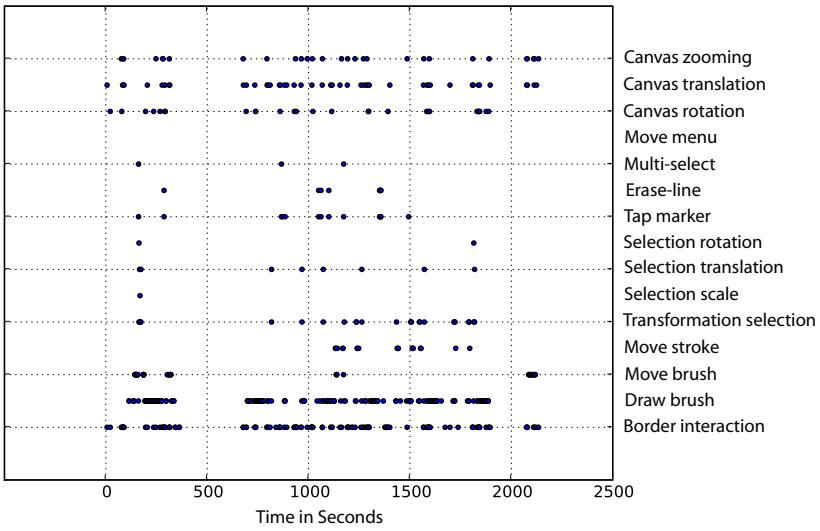


Figure 44: The command execution log of the case study sketch.

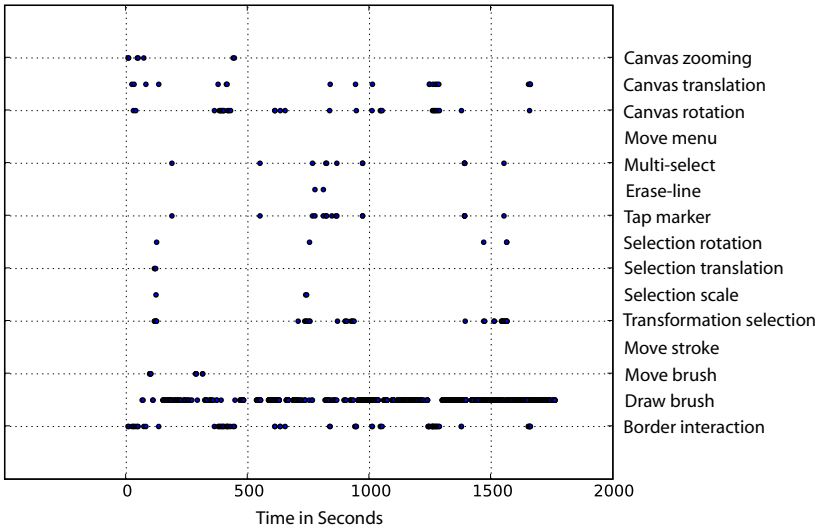


Figure 45: The command execution log of a participant during the evaluation.

The usage of the other interactions is far more situational. The move brush and move stroke are occasionally used to correct mistakes or change existing lines. The tap marker is used more frequently as it ties

in to three distinct interactions, although the option to move the menu with the marker is hardly used at all. This verifies the low popularity of that particular interaction. There is some indication that the frame gestures take a while to fully adapt to. The artist responsible for the log of Figure 44 had worked with the implementation significantly longer than the artist responsible for Figure 45. This artist, who had more experience working with the software, uses the gestures far more frequently than the artist who had only just started experimenting. Still, despite being new to the system, the artist was already using the frame gestures often, indicating that the cost of learning and using the gestures is low.

5.4 RESULTS

As this thesis is ultimately about being able to create art, it would not be complete without some examples of the artwork created through the research which has been presented. Here are some of the images that were made using the implementation (Figure 46 to Figure 52).

Figure 46: Image courtesy of Avik Kumar Maitra.

Figure 47

Figure 48: Image courtesy of Sahal Merchant.

Figure 49

Figure 50

Figure 51: Image courtesy of Sahal Merchant.

Figure 52

5.5 SUMMARY

Through the case study we have shown how the interactions presented in this thesis can be used. The case study itself was a time lapse of a sketch, highlighting the use of each interaction. The resulting sketch showed that the implementation can be used to create art.

The informal user evaluation gave a lot of insight into the way people react to the implementation. The participants were divided into two groups: artists and non-artists. Opinions and suggestions from both groups were remarkably similar. Overall, the artist group rated the implementation slightly lower than the non-artists. The most successful interaction was by far the frame gestures. Participants found them a useful addition to a drawing system. The move brush was also well liked. Participants were less pleased with the move stroke, page corner saving and method by which the menu was moved. The most common suggestions by participants were the inclusion of an undo button and tooltips for the menu.

To see how the interactions were used during the sketching process we analyzed several command execution logs. These showed that drawing and frame gestures were used frequently and consistently throughout the sketching process. Most of the other interactions were more situational, their usage depending on what the artist wanted to draw.

We ended the chapter with examples of sketches that were made by participants, using the implementation. These sketches also show that the implementation can accommodate a wide range of styles, despite being more suitable to some styles than others.

6 CONCLUSION AND FUTURE WORK

The goal of this thesis was to explore new ways in which artists can sketch digitally and interact with strokes. To accomplish this, a new look was taken at some of the basic drawing interactions which artists frequently use. Also, new types of interactions were researched. An important factor in achieving this goal was to find ways to perform interactions that did not disturb the workflow of the artist.

The work was motivated by the growing popularity of digital art creation. Hardware such as pen tablets are becoming more accessible to artists and digital work comes with certain advantages. Storing digital paintings requires no physical space and artists need not concern themselves with, e. g. paint mixing and turpentine fumes. However, many popular drawing systems have seen little innovation over the last few decades concerning the way artists can interact with their digital work. With the advancements in ergonomics and computing power since then, it is a good time to re-evaluate these interactions, and possibly design improved ways in which to perform them.

To reach the goal set in this thesis, several interaction concepts were researched. The concepts broadly fall into two categories: canvas interactions and stroke interactions. Canvas interactions are meant to allow the artist to use the canvas as a tool. Giving free control over the canvas allows the artist to orient it in a way which allows him to draw in the most comfortable way. As such, the canvas assists the artist in the drawing process. Stroke interactions were developed to give the artist more control over lines after they have been drawn, in an intuitive way. In order to interrupt the workflow of the artist as little as possible, we focused on modeless ways in which these interactions could be done. It was also a priority to keep the interface clean, simple and visually pleasing.

The canvas interactions were implemented as a set of location sensitive gestures. This allows the artist to orient the canvas through simple motions: translation, scaling and rotation are all supported. This method of interaction is also modeless, as no buttons or key combinations need to be pressed to be able to use the interactions. A similar scheme was used to control the stroke selection border. To extend the set of commonly used stroke interactions, the move brush was developed. This brush gives the artist an intuitive tool to manipulate the shape of strokes, in a familiar feeling way. In an attempt to generalize the move brush, the move stroke was presented. The move stroke works like the move brush, only now the artist can define his or her own brush from any stroke on the canvas. Finally, the tap marker was introduced. This allowed for several interactions to be performed using the tap gesture. The tap marker was used to perform the multi-select and erase-line interactions, as well as opening up the possibility to move the menu.

The evaluation showed that the frame gestures were very popular with all participants. Most found it a valuable addition to the interactions commonly available in digital drawing systems. The move brush was found by many to be an interesting tool to experiment with. Less liked by the participants were the way in which the menu could be moved, the page flipping concept and the execution of the move stroke. Some suggestions were given to improve the implementation. The most common suggestions were an undo button and tooltips for the menu. An analysis of several command execution logs showed that the frame gestures are used frequently and consistently. They seem to integrate well with the workflow of different artists. The usage of many interactions depend on what the artist is drawing and what his particular style is. This shows that the use of the move brush and transformation selection are much more situational.

The concepts presented in this thesis opens up several interesting research directions. The frame gestures give artists a new way to manipulate the canvas. Perhaps there are other interesting ways in which gestures can be used to help the artist accomplish certain tasks. Also, no formal study was done on the effect frame gestures have on the artists workflow and efficiency. In order to truly understand what merit these gestures have, such a study would have to be undertaken.

The move brush, while effective, is just a basic implementation of what the concept could be. Possible extensions could be a more flexible brush model or a way to texture the brush. The move stroke concept also warrants a second look. While not very successful in its current incarnation, the idea of defining your own brushes in such a simple and intuitive manner is a very powerful concept.

An important feature the current stroke model lacks, is a way to define the texture or style of strokes. Extending the model to accommodate stroke textures or to simulate different media could prove a worthwhile endeavor. While the current stroke model is fairly robust, there are still cases where it is not accurate enough. Continued development would have to be performed in order to make it as reliable as the strokes of pixel based systems. Also, efficiency was not a high priority for this thesis, so there are still many areas of the model which would benefit from optimization.

Regarding the interface, while the absence of modes appealed to all who used the implementation, many argued that restricting the interface to only a pen tablet hampered their workflow. It would be interesting to research the exact dynamics of user input devices for digital painting, in order to find the optimal combination.

An area that can use significant improvements is the tap interaction. In order to make the multi-selection and erase-line interactions more useful and streamlined, they could use the stroke paradigm: the artist draws an (arbitrary) stroke from the tap marker which intersects several sketched lines. When the artist is finished with the interaction, these lines are selected (and possibly erased). This eliminates the conceptual differences between both interactions and possibly makes them more flexible at the same time. This approach is inspired by CrossY [Apitz

and Guimbretière, 2005]. The tap could be improved by changing the time needed for the stylus to touch the tablet in order to make the tap marker appear. Currently, this contact time is very short. Making this time relatively long might work better with some drawing styles, while not having an adverse effect on other styles.

Yet another interesting direction to explore is to make the interface work with (bimanual) touch interaction. The current system was designed with large displays in mind, and the interactions can be extended to work with this type of interaction. Bimanual interaction has become a popular topic in recent years, and researching effective ways in which it could be used to draw could prove valuable.

As both an artist and a computer science student, this thesis was a wonderful opportunity to combine two of my greatest interests. Doing the research has made me look more critical at my own artistic process, in both drawing and programming. That, combined with the inspiration I have found through working with all the wonderful people, who in some way contributed to this thesis, have motivated me to become a better artist, and through that a better computer scientist.

BIBLIOGRAPHY

Adobe. Illustrator, 1988. http://www.adobe.com/products/illustrator/. (Cited on pages 5, 6, and 28.)

Adobe. PhotoShop, 1990. http://www.adobe.com/products/photoshop/family/. (Cited on pages 1, 6, 9, and 20.)

Marc Alexa, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin, and Claudio T. Silva. Point set surfaces. In *VIS '01: Proceedings of the conference on Visualization '01*, pages 21–28. IEEE Computer Society, 2008. (Cited on page 37.)

AmbientDesign. ArtRage, 2004. http://artrage.com/. (Cited on pages 6 and 19.)

Georg Apitz and François Guimbretière. CrossY: a crossing-based drawing application. In *SIGGRAPH '05: SIGGRAPH 2005 Papers*, pages 930–930. ACM, 2005. (Cited on pages 51 and 62.)

Caroline Appert, Olivier Chapuis, and Michel Beaudouin-Lafon. Evaluation of pointing performance on screen edges. In *AVI '08: Proceedings of the working conference on Advanced visual interface*, pages 119–126. ACM, 2008. (Cited on page 22.)

Seok-Hyung Bae, Ravin Balakrishnan, and Karan Singh. ILoveSketch: as-natural-as-possible sketching system for creating 3d curve models. In *SIGGRAPH '06: SIGGRAPH 2006 Papers*, pages 151–160. ACM, 2008. (Cited on pages 8, 19, 21, and 22.)

Ravin Balakrishnan and Ken Hinckley. Symmetric bimanual interaction. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 33–40. ACM Press, 2000. (Cited on page 16.)

Pascal Barla, Joëlle Thollot, and Françis X. Sillion. Geometric clustering for line drawing simplification, 2005. (Cited on page 13.)

Olivier Bau and Wendy E. Mackay. OctoPocus: a dynamic guide for learning gesture-based command sets. In *Proceedings of the 21st annual ACM symposium on User interface software and technology*, pages 37–46. ACM Press, 2008. (Cited on pages 15 and 21.)

O. Bottema. On the area of a triangle in barycentric coordinates. *Crux Mathematicorum*, 8:228–231, 1982. (Cited on page 43.)

J. Bruijns. Quadratic bezier triangles as drawing primitives. In *HWWS '98: Proceedings of the ACM SIGGRAPH/EUROGRAPHICS workshop on Graphics hardware*, pages 15–25. ACM, 1998. (Cited on page 39.)

William Buxton and Brad A. Myers. A study in two-handed input. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 321–326. ACM Press, 1986. (Cited on page 16.)

Stéphane Chatty, Stéphane Sire, Jean-Luc Vinot, Patrick Lecoanet, Alexandre Lemort, and Christophe Mertz. Revisiting visual interface programming: creating gui tools for designers and programmers.

In *UIST '04: Proceedings of the 17th annual ACM symposium on User interface software and technology*, pages 267–276. ACM, 2004. (Cited on page 19.)

Jin J. Chou and Les A. Piegl. Data reduction using cubic rational b-splines. *IEEE Computer Graphics and Applications*, 12(3):60–68, 1992. (Cited on page 33.)

Corel. Corel Painter, 2007. http://www.corel.com/. (Cited on pages 5, 6, and 19.)

Eric Daniels. Deep canvas in Disney's Tarzan. In *International Conference on Computer Graphics and Interactive Techniques ACM SIGGRAPH 99 Conference abstracts and applications*, page 200, New York, NY, USA, 1999. ACM Press. (Cited on page 9.)

Doug DeCarlo and Szymon Rusinkiewicz. Highlight lines for conveying shape. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, pages 63–70. ACM Press, 2007. (Cited on page 13.)

Debra Dooley and Michael F. Cohen. Art-based rendering of fur, grass, and trees. In *Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 77–82. ACM Press, 1999. (Cited on page 12.)

W. S. Dorn. A generalization of horner's rule for polynomial evaluation. In *Proceedings of the 1961 16th ACM national meeting*, pages 501–502. ACM, 1961. (Cited on page 34.)

Frédo Durand, Victor Ostromoukhov, Mathieu Miller, François Duranleau, and Julie Dorsey. Decoupling strokes and high-level attributes for interactive traditional drawing. In *Proceedings of the 12th Eurographics Workshop on Rendering Techniques*, pages 71–82. Springer-Verlag, 2001. (Cited on page 8.)

Jihad El-Sana, Elvir Azanli, and Amitabh Varshney. Skip strips: maintaining triangle strips for view-dependent rendering. In *Proceedings of the conference on Visualization '99: celebrating ten years*, pages 131–138. IEEE Computer Society Press, 1999. (Cited on page 35.)

Gershon Elber. Line illustrations in computer graphics. *The Visual Computer*, 11(6):290–296, August 1995. (Cited on page 12.)

George W. Fitzmaurice, Ravin Balakrishnan, Gordon Kurtenbach, and Bill Buxton. An exploration into supporting artwork orientation in the user interface. In *CHI '99: Proceedings of the SIGCHI conference on Human factors in computing systems: the CHI is the limit*, pages 167–174. ACM, 1999. (Cited on page 21.)

Clifton Forlines, Daniel Wigdor, Chia Shen, and Ravin Balakrishnan. Direct-touch vs. mouse input for tabletop displays. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 647–656. ACM Press, 2007. (Cited on page 16.)

Luigi Gallo, Giuseppe De Pietro, and Ivana Marra. 3D interaction with volumetric medical data: experiencing the Wiimote. In *Proceedings of the 1st international conference on Ambient media and systems*. ICST, 2008. (Cited on page 17.)

Stéphane Grabli, Emmanuel Turquin, Frédo Durand, and François Sillion. Programmable style for NPR line drawing. In *Rendering Techniques 2004 (Eurographics Symposium on Rendering*. ACM Press, 2004. (Cited on page 13.)

Tovi Grossman, Ravin Balakrishnan, and Karan Singh. An interface for creating and manipulating curves using a high degree-of-freedom curve input device. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 185–192. ACM Press, 2003. (Cited on page 17.)

Jens Grubert, Sheelagh Carpendale, and Tobias Isenberg. Interactive stroke-based NPR using hand postures on large displays. In *Short Papers at Eurographics 2008*, pages 279–282. Eurographics Association, 2008. (Cited on pages 14 and 15.)

Sunil Hadap, Dave Eberle, Pascal Volino, Ming C. Lin, Stephane Redon, and Christer Ericson. Collision detection and proximity queries. In *SIGGRAPH '04: SIGGRAPH 2004 Course Notes*. ACM, 2004. (Cited on page 43.)

Paul Haeberli. DynaDraw. `http://www.graficaobscura.com/dyna/index.html`, 1989. (Cited on pages 6 and 33.)

Paul Haeberli. Paint by numbers: abstract image representations. *Communications of the ACM*, 24(4):207–214, August 1990. (Cited on pages 5 and 6.)

Aaron Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 453–460. ACM Press, 1998. (Cited on page 11.)

Aaron Hertzmann, Nuria Oliver, Brian Curless, and Steven M. Seitz. Curve analogies. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 233–246. Eurographics Association, 2002. (Cited on page 10.)

Siu Chi Hsu and Irene H. H. Lee. Drawing and animation using skeletal strokes. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 109–118, New York, NY, USA, 1994. ACM Press. (Cited on pages 9, 10, 11, and 14.)

Tobias Isenberg, André Miede, and Sheelagh Carpendale. A buffer framework for supporting responsive interaction in information visualization interfaces. In *Proceedings of the Fourth International Conference on Creating, Connecting and Collaborating through Computing*, pages 262–269. IEEE Computer Society, 2006. (Cited on page 31.)

Vinit Jain and Ben Shneiderman. Data structures for dynamic queries: an analytical and experimental evaluation. In *AVI '94: Proceedings of the workshop on Advanced visual interfaces*, pages 1–11. ACM, 1994. (Cited on page 44.)

Robert D. Kalnins, Lee Markosian, Barbara J. Meier, Michael A. Kowalski, Joseph C. Lee, Philip L. Davidson, Matthew Webb, John F. Hughes, and Adam Finkelstein. WYSIWYG NPR: drawing strokes directly on 3D models. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 755–762. ACM Press, 2002. (Cited on page 8.)

Daniel F. Keefe, Daniel Acevedo Feliz, Tomer Moscovich, David H. Laidlaw, and Jr. Joseph J. LaViola. CavePainting: a fully immersive 3D artistic medium and interactive experience. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 85–93, New York, NY, USA, 2001. ACM Press. (Cited on pages 7 and 17.)

Michael A. Kowalski, Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barze, Loring S. Holden, and John F. Hughes. Automatic illustration of 3D geometric models: lines. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pages 433–438. ACM Press, 1999. (Cited on page 12.)

Dustin Lang, Leah Findlater, and Michael Shaver. CoolPaint: direct interaction painting, 2003. (Cited on page 7.)

Yunjin Lee, Lee Markosian, Seungyong Lee, and John F. Hughes. Line drawings via abstracted shading, 2007. (Cited on page 12.)

J. P. Lewis, Nickson Fong, Xie XueXiang, Seah Hock Soon, and Tian Feng. More optimal strokes for NPR sketching. In *Proceedings of the 3rd international conference on Computer graphics and interactive techniques in Australasia and South East Asia*, pages 47–50. ACM Press, 2005. (Cited on page 13.)

Kyoko Murakami, Reiji Tsuruno, and Etsuo Genda. Multiple illuminated paper textures for drawing strokes. In *Proceedings of the Computer Graphics International 2005*, pages 156–161. IEEE Computer Society, 2005. (Cited on pages 10 and 11.)

Nintendo. Wiimote, 2007. `http://wii.nintendo.com/`. (Cited on pages 16 and 17.)

QtSoftware. Qt, 1995. `http://www.qtsoftware.com/`. (Cited on page 31.)

Ari Rappoport. Rendering curves and surfaces with hybrid subdivision and forward differencing. *Transactions on Graphics (TOG)*, 10(4):323–341, 1991. (Cited on page 35.)

Dean Rubine. Specifying gestures by example. In *Proceedings of the 18th annual conference on Computer graphics and interactive techniques*, pages 329–337. ACM Press, 1991. (Cited on pages 15 and 21.)

Thomas Schlömer, Benjamin Poppinga, Niels Henze, and Susanne Boll. Gesture recognition with a Wii controller. In *Proceedings of the 2nd international conference on Tangible and embedded interaction*, pages 11–14. ACM Press, 2008. (Cited on page 17.)

Hock Soon Seah, Zhongke Wu, Feng Tian, Xian Xiao, and Boya Xie. Artistic brushstroke representation and animation with disk b-spline curve. In *Proceedings of the 2005 ACM SIGCHI International Conference on Advances in computer entertainment technology*, pages 88–93, New York, NY, USA, 2005. ACM Press. (Cited on pages 9 and 34.)

A.R. Smith. Digital paint systems: an anecdotal and historical overview. *IEEE Annals of the History of Computing*, 23(2):4–30, August 2001. (Cited on page 5.)

Sara L. Su, Ying-Qing Xu, Heung-Yeung Shum, and Falai Chen. Simulating artistic brushstrokes using interval splines. In *Proceedings of the 5th International Conference on Computer Graphics and Imaging*, pages 85–90, August 2002. (Cited on pages 10 and 34.)

Peter Vandoren, Tom Van Laerhoven, Luc Claesen, Johannes Taelman, Chris Raymaekers, and Frank Van Reeth. IntuPaint: Bridging the gap between physical and digital painting. In *Proceedings of the Third Annual IEEE International Workshop on Horizontal Interactive Human-Computer Interaction*, pages 71–78. IEEE Computer Society, 2008. (Cited on page 7.)

Wacom. http://www.wacom.com/. (Cited on pages 1 and 16.)

Ingo Wald, Thiago Ize, Andrew Kensler, Aaron Knoll, and Steven G. Parker. Ray tracing animated scenes using coherent grid traversal. In *SIGGRAPH '06: SIGGRAPH 2006 Papers*, pages 485–493. ACM, 2006. (Cited on page 44.)

Huagen Wan, Yang Luo, Shuming Gao, and Qunsheng Peng. Realistic virtual hand modeling with applications for virtual grasping. In *Proceedings of the 2004 ACM SIGGRAPH international conference on Virtual Reality continuum and its applications in industry*, pages 81–87. ACM Press, 2004. (Cited on page 17.)

Mike Wu and Ravin Balakrishnan. Multi-finger and whole hand gestural interaction techniques for multi-user tabletop displays. In *Proceedings of the 16th annual ACM symposium on User interface software and technology*, pages 193–202. ACM Press, 2003. (Cited on pages 14 and 15.)

Xara. Xara X. http://www.xara.com/. (Cited on page 6.)

DECLARATION

I declare that this thesis has been composed by myself, that the work contained herein is my own except where explicitly stated otherwise in the text, and that this work has not been submitted for any other degree or professional qualification except as specified. Furthermore, I declare that this thesis may be used for publication in collaboration with my supervisors, Dr. Tobias Isenberg and Moritz Gerl.

Groningen, February 2009

Menno Nijboer