Physically-Based Modeling Techniques for Interactive Digital Painting

by
William Valentine Baxter III

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2004

Approved by:

Ming C. Lin, Advisor

Dinesh Manocha, Reader

Gary Bishop, Reader

Anselmo Lastra, Committee Member

Michael Minion, Committee Member

ABSTRACT

WILLIAM VALENTINE BAXTER III: Physically-Based Modeling Techniques for Interactive Digital Painting
(Under the direction of Ming C. Lin)

In this dissertation I present a novel, physically-based approach to digital painting. With the interactive simulation techniques I present, digital painters can work with digital brushes and paints whose behavior is similar to real ones. Using this physically-based approach, a digital painting system can provide artists with a versatile and expressive creative tool, while at the same time providing a more natural style of interaction enabled by the emulation of real-world implements.

I introduce several specific modeling techniques for digital painting. First, I present a physically-based, 3D, deformable, virtual brush model based on non-linear quasi-static constrained energy minimization. The brush dynamics are computed using a skeletal physical model, which then determines the motion of a more complex geometric model. I also present three different models for capturing the dynamic behavior of viscous paint media, each offering a different trade-off between speed and fidelity—from 2D heuristics, to 3D partial differential equations. Accurate modeling of the optical behavior of paint mixtures and glazes is also important, and for this I present a real-time, physically-based rendering technique, based on the Kubelka-Munk equations and an eight-sample color space. Finally, I present techniques for modeling the haptic response of brushes in an artist's hand, and demonstrate that all these techniques can be combined to provide the digital painter with an interactive, virtual painting system with a working style similar to real-world painting.

To Papa Bear,

a.k.a. William V. Baxter, Sr.

ACKNOWLEDGMENTS

It takes a tremendous amount of dedication and perseverance to complete a doctoral dissertation. At least that's what I've been told. I'm not sure I possess either of those qualities in great enough quantity to complete a dissertation, but I am fortunate in that I have been constantly surrounded by many great and motivated people whose encouragement and assistance throughout this process has made up for any attributes I lack. I would like to sincerely extend my gratitude to those individuals here.

I would first like to thank my dear friend Leo Chan for setting me off on this course. I got together with Leo sometime in the fall of 1997, and he mentioned to me that he was working on computer graphics. It was his compelling description of a field combining computer science, math, and physics, with elements of visual design and art that piqued my interest and led me to apply to graduate schools.

Thanks to Mark Harris and Vincent Scheib, for good discussions and inspirations relating to research topics over the years, but also for inspiration to greater heights in every form of technical and non-technical communication. Thanks as well to Kenny Hoff for his infectious enthusiasm about graphics and many stimulating discussions on that and any number of other topics.

I extend my humble appreciation to Greg Coombe, Jeff Feasel, and Karl Gyllstrom for letting me make noise with them, the one endeavor that may have done the most to preserve my sanity in the last couple of years.

Thanks to all the painters who used and tested my painting systems: Rebecca Holmberg, John Holloway, Andrea Mantler, Haolong Ma, Sarah Hoff, my wife Eriko Baxter, Lauren Adams from the Art department, and all those who gave me valuable feedback after seeing the demo. An especially large thanks to painter John Holloway for his enthusiasm, encouragement, and belief in this project, and to Rebecca Holmberg for going far beyond the call of duty to create so many delightful paintings with dAb while at the same time finishing her own degree in chemistry. I wish I could do more, but all I have to offer is my sincere thanks and an honorary degree from the "Baxter school of digital painting."

I am very grateful for all the help and the late nights given up by collaborators on this and previous research: Avneesh Sud, Naga Govindaraju, Vincent Scheib, Yuanxin Liu, and Jeremy Wendt. And thanks to those who allowed me to collaborate with them on their research as I learned the ropes: Carl Erikson, and the late-90's Walkthrough MMR all-star gang – Dan Aliaga, Rui Bastos, Jon Cohen, Dave Luebke, Andy Wilson, and Hansong Zhang. I would especially like to thank Vincent Scheib for his help on the dAb paint model, and Jeremy Wendt for his invaluable assistance in the paint measurement effort and with the Kubelka-Munk rendering code.

Thanks to Dinesh Manocha for giving me the opportunity to work with him and the Walkthrough group for my first two and a half years of graduate school. Without his willingness to take a chance on me, a kid with no computer graphics experience, I would probably still be shoveling business logic code around today.

Many thanks to my advisor, Ming Lin, whose firm belief in my abilities and belief in this topic kept me going. She is tough on her students, but her tireless promotion of her students' interests behind the scenes does not go unnoticed. Thanks to Dr. Lin and Dr. Manocha's support, I have always been free to pursue the research ideas I wanted to pursue throughout my time here.

Thanks also to everyone who read this admittedly long dissertation, and for their invaluable feedback. I am especially grateful to Ming Lin for working with me throughout the writing process, and to my outside readers, Gary Bishop and Dinesh Manocha. Also thanks to my other committee members, Anselmo Lastra and Michael Minion, who also took the time to read large portions and provide constructive feedback. Their efforts greatly improved this dissertation.

I would like to express my appreciation also to the the agencies and foundations that have provided support for my work. First to the Link Foundation and NVIDIA for fellowships in my last two years, and to the agencies who have sponsored various research in the GAMMA group: Intel Corporation, the National Science Foundation, the Office of Naval Research, and the U.S. Army Research Office.

Also, a big thanks to my parents, Suzanne and Bill, Jr., for their love and support from the beginning and especially during this time. Three days before my defense I began to find out just how much I really owe them when I became a father myself. Thanks to my sister Amy for breaking ground by becoming the first (but now not the only!) doctor in the family and for always inspiring me to try harder throughout my life.

Finally, I would like to thank to my wife, Eriko, for her tireless dedication, for the sacrifices she has made, for always staying by my side, for setting a shining example with her seemingly bottomless persistence, for her unwavering belief in the value of seeing the dissertation through to the end, and for all the encouragement she provided in the times I felt hopeless. Thank you, Elly-chan.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

Chapter 1

Introduction

We don't make mistakes here, we just have happy accidents. We want happy, happy paintings. If you want sad things, watch the news. Everything is possible here. This is your little universe.
— Bob Ross, on his popular PBS show, "The Joy of Painting".

Painting is a universally appreciated and widely practiced form of art with a magnificent history. Nearly every culture across the globe regards some form of painting among its primary arts. Paintings found on the walls of caves in Europe dating back to 40,000 B.C. (Johnson, 2003) attest to the long-standing fascination of humans with painting (Figure 1.1). This fascination is as strong today as ever, with hundreds of museums across the world displaying their painting collections to millions of visitors every year. It is impossible to put a price on the value of painting to mankind, but the recent sale of a painting for $104 million[1] certainly gives an indication of painting's staggering value to man.

The enduring popularity of painting is probably due to a combination of its simplicity and range. It is something simple enough that a child can do it, yet deep enough to require a lifetime to master. Painting is among the more versatile of the visual arts. In a painting, an artist is able to give manifestation to almost any imaginative impulse,

[1]Picasso's *Garçon a la Pipe* sold for a record price of $104 million in May, 2004. Prior to that the record was $82.5 million, paid for Vincent Van Gogh's *Portrait of Dr. Gachet* in 1990. (Source: BBC News, http://news.bbc.co.uk/2/hi/entertainment/3682127.stm)

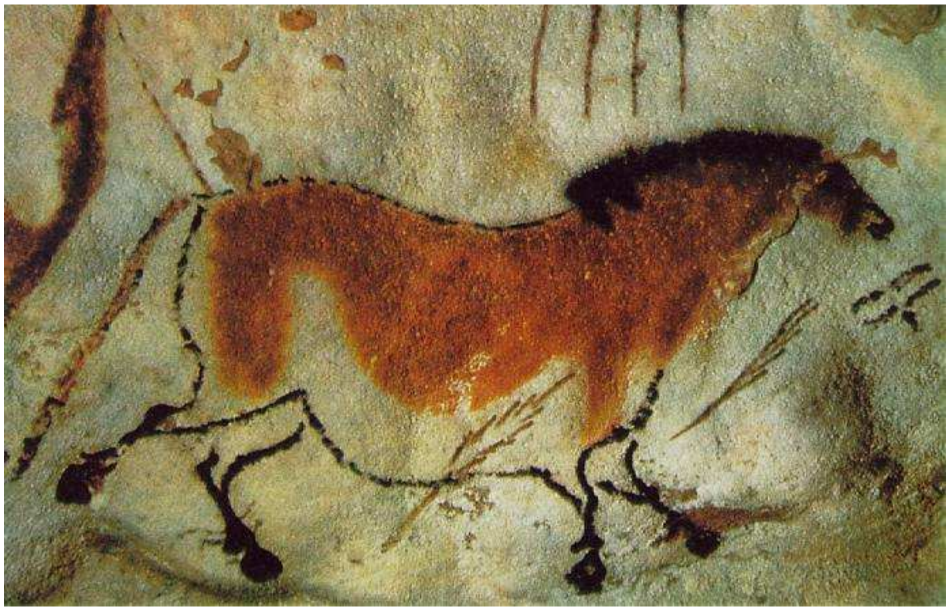


Figure 1.1: *An ancient cave painting, c. 15,000-10,000 B.C., located in Lascaux, France. (Scan by Mark Harden, http://www.artchive.com)*

no matter how far removed from reality. A good painting has the ability to draw the viewer in, to tell a story, to transport the viewer to another location or into another state of mind, to alter mood, to provoke thought, or to inspire action. Paintings can be convincingly lifelike (Figure 1.2), or viscerally abstract (Figure 1.3); they can be spacious and expansive (Figure 1.4), or closed and personal (Figure 1.5). The variety and range of paintings is limitless.

1.1 Painting and Computers

We do not typically think of painting as technology. However, in the days of Leonardo da Vinci, the pursuit of improved painting technique was the very pursuit of advanced imaging technology. Da Vinci and others sought to discover the principles necessary to create realistic paintings, just as computer graphics researchers since the 1970's have sought to develop computer algorithms for realistic image synthesis. Throughout the majority of human history, painting was the most technologically sophisticated means of

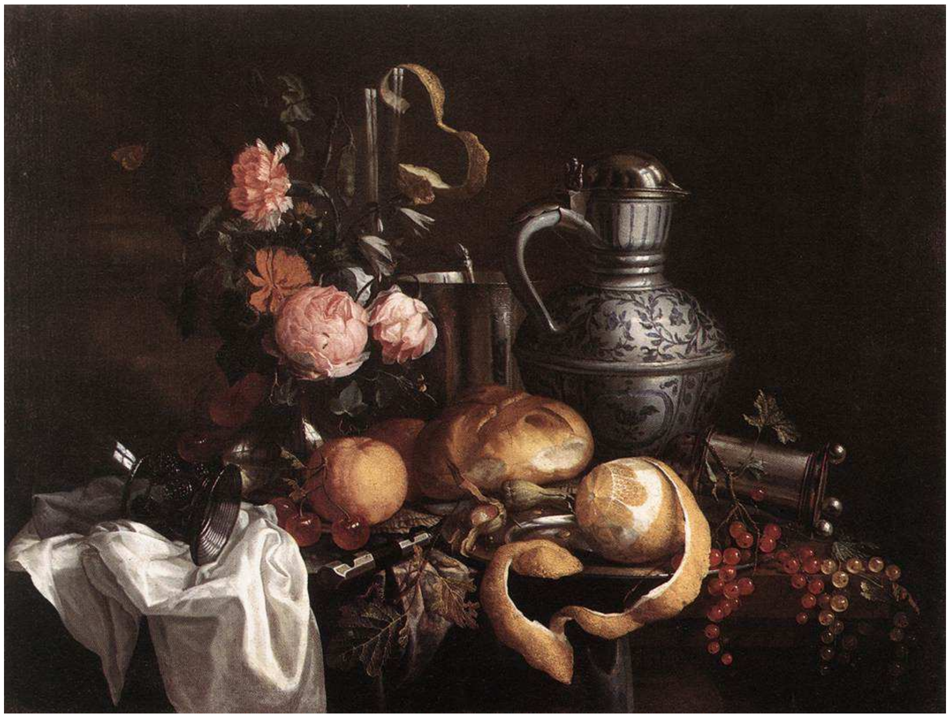


Figure 1.2: *Still-Life, Jan Davidsz de Heem, c. 1650*

image reproduction. This remained so up until about 1837 when Louis-Jacques-Mandé Daguerre introduced the world to photography(Honour & Fleming, 1995). Today, when one thinks of imaging technology, one looks towards computer graphics.

Modern computer graphics have enabled some truly stunning artistic creations that would be impossible to achieve any other way. Da Vinci, though a genius, could never have painted Toy Story[2]. Yet it was with nothing but the traditional tools of painting that he and other great masters over the ages were able to create many stunningly realistic and lasting masterpieces. These great artists—Rembrandt, Goya, Vermeer, and countless others—accomplished their feats with the help of the tools and materials at hand. And their tools were essentially the same tools that a student of painting would use today. Centuries of technological advancement have not altered the basic process of painting, simply because that process is highly effective.

[2]Pixar's—and the world's—first feature length 3D computer animated film.

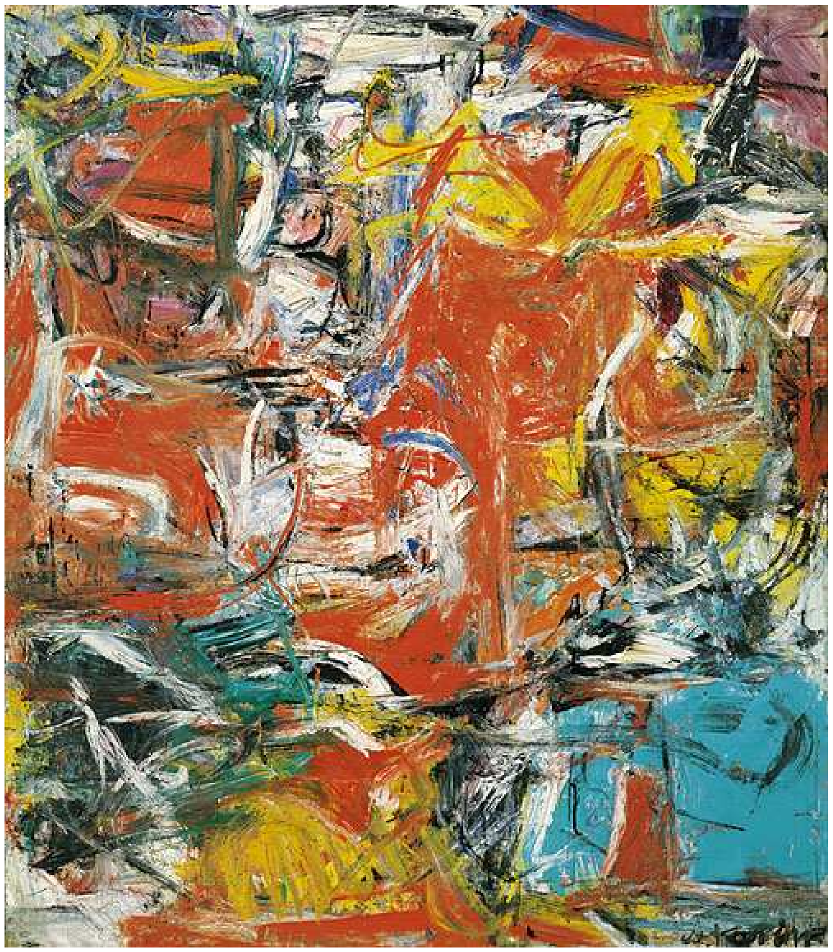


Figure 1.3: *Composition, Willem de Kooning, 1955. Solomon R. Guggenheim Museum. 55.1419. ©2003 The Willem de Kooning Foundation/Artists Rights Society (ARS), New York.*

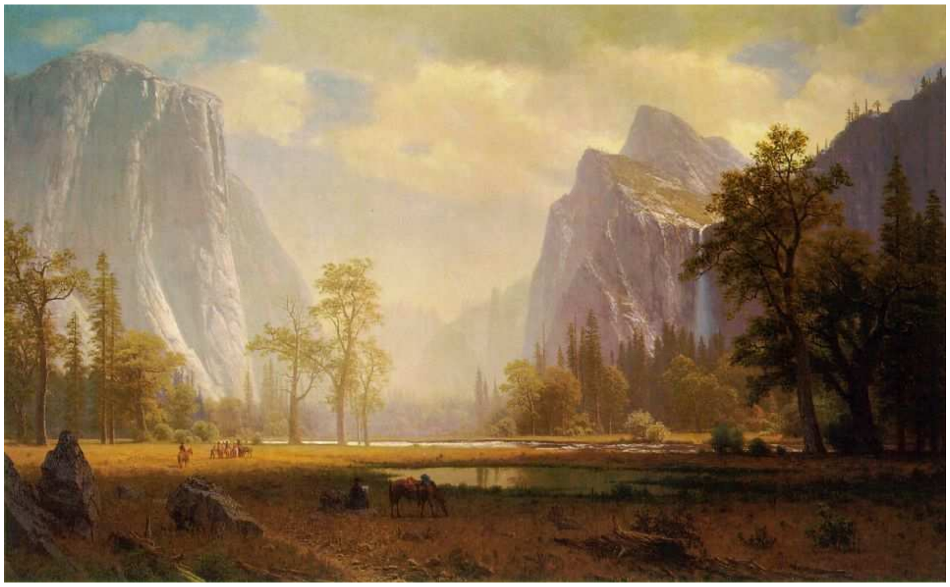


Figure 1.4: *Looking Up the Yosemite Valley, Albert Bierstadt, c. 1865-67 (Scan by Mark Harden, http://www.artchive.com)*

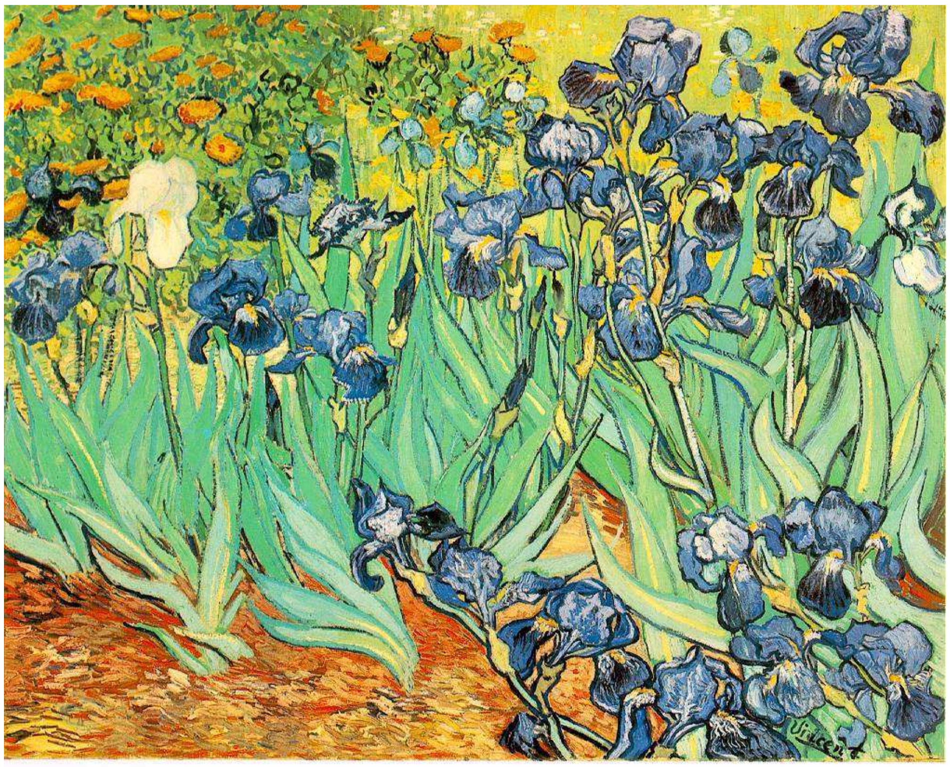


Figure 1.5: *Irises, Vincent Van Gogh, 1889. (Scan by Mark Harden, http://www.artchive.com)*

Computers have changed most artists' lives relatively little, but at the same time, the rise of computers has significantly impacted the lives of others. Writers have widely adopted word processors as the main tool of their craft. Today, very few writers or students create works of significant length with pen and paper. Computers have also transformed many engineering tasks such as design and manufacturing. The same transformation has occurred in the worlds of audio and video production. But, in contrast, most visual artists today still work with traditional materials. Relatively few have embraced computer-based drawing and painting tools for serious artwork.

Why have artists not adopted computers more widely? It is not out of ignorance of the many advantages the computer can offer, like undo and version control. The problem, many artists argue, is that computer images lack a certain human warmth. (See (Cockshott, 1991), and Appendix B, e.g.) Tunde Cockshott, an artist by training, argued in his dissertation that computer painting programs he had tried all "took far too shallow a view of the process of real painting," going on to cite the common opinion among artists (including himself) that these programs were all "too sterile" (Cockshott, 1991, p.10). Artist John Holloway agrees, saying most computer generated images lack what he calls "organicness" and are "too clean, too sterile". (Holloway has been working with my IMPaSTo painting system for some time. His full comments appear in Appendix B.)

In order for computer images to shed this sterility, those who create computer painting systems must understand how artists view painting. There are three closely related aspects of real painting that are important to artists, but that are typically overlooked by painting programs. These are the importance of the *process*, the importance of *accident*, and the importance of *complexity*. In order to create painting systems that artists will embrace, it is necessary to understand these three concepts and arrange for them to be fundamental to the operation of the digital system.

Painting is a process. It is a fusion of feeling and action, sight and touch, purpose and paint, which goes beyond merely producing an image to give an artistic impression (Mayer, 1991). Current painting programs focus more on the product, on the final image, rather than the process of getting there. David Em comments that this emphasis is typical of the engineering mindset, to put the product before the process, whereas the process comes first with the artist when making a work of art (Em, 1983). Cockshott writes about the iterative and interactive nature of the painting process (Cockshott, 1991, p.21), how the artist starts with an idea, and moves to achieve that end, but ultimately “cannot but help be influenced by the *process* of making the image” (emphasis his). Computer artist Lillian Schwartz writes about the process becoming encapsulated in the image. She says that if the artist is effective “we remain moved because of the spirit preserved in the artwork” (Schwartz, 1992). Current painting programs focus too much on creating eye-catching effects with a single click, to the detriment of the overall process of painting.

Closely related to process is the concept of the *accidental.* The accident, in some sense, gives rise to process in creating art. The artist tries something, sees what results, decides where to go from there, and repeats this cycle until it seems that any further attempt at improvement can only detract. Often “what results” is not what the artist expected, and this can lead him or her in new directions. Many artists have talked about the importance of the accidental in art. Pablo Picasso embraced the accidental, summing up his artistic process glibly as: “I begin with an idea and then it becomes something else.” He believed in the importance of the idea merely as a starting point (Ashton, 1972, p.72). Jackson Pollock writes about the unpredictable direction a painting inevitably takes during its creation, describing a painting as something with “a life of its own” (Chipp, 1968, p.584). Popular painting guru Bob Ross believed in the benefit of accidents as well, as is plain from the quote at the beginning of this chapter, and his frequent use of the phrase “happy accident”. Most painting programs go out of their way to make every operation completely predictable and precise, leaving little room for the type of spontaneity that gives rise to accident.

Finally, the role of *complexity* is important as well. Real painting materials have a rich, dynamic behavior, and not even a master painter can say for sure what a mark is going to look like until the brush finally leaves the canvas. At the same time, the mark made is also not completely random. The skilled painter’s exerted control will almost always achieve the general, if not the specific, outcome desired. (Lewis, 1984) tries to give computer painting more organic complexity by adding fractal noise to stroke textures. Though he succeeds in adding complexity, the control aspect is neglected, leading to marks that have complexity largely uncorrelated with the painter’s actions. The richness of natural material behavior leads to a richness of expression in the resulting work that is, nonetheless, still under artist control. Complexity relates back to the notion of the accidental as well. More complexity naturally leads to an increased potential for accident. This connection is expressed vividly in the drip and splatter works of Jackson Pollock. (Cockshott, 1991, p.25) criticizes the brushes provided by typical painting programs, arguing they are created by feature-happy engineers who know little about what artists want. He writes,

> the role of the artistically-naive developer of computer-based paint systems has been perpetuated. This is evident in the range of tools and effects available in the contemporary commercial products, many of which have little real value other than as gimmicks such as a brush which paints a trail of musical notes.

He further derides other, less gimmicky types of brushes offered by these programs as being no more expressive than a simple child’s potato stamp (Cockshott, 1991, p.26). There is little chance for accident when the tools at hand are imbued with inhuman digital precision. In contrast, the materials and tools relied on by artists are rich with natural complexity. Brushes are composed of thousands of individual hairs, each able to move independently. Paint squishes, flows, adheres, and spreads in often unpredictable ways. Even in the hand of an expert, a brush will never make exactly the same mark twice. The effect of this complexity is not pure randomness, but more a steerable unpredictability. The artist still has the ultimate control.

In this thesis I propose new models of the traditional painting materials and tools that enable the creation of painting systems that offer all three of these important aspects to the artist—process, accident, and complexity.

1.2 Painting Media

Very little has fundamentally changed in the craft of painting from the days of the first painters. The basic challenge remains the same: "to fix pigments to a ground in order to preserve them." (Honour & Fleming, 1995). This is accomplished by preparing a support, such as wood, rock, or canvas, with a film of ground onto which the pigment can adhere. What has changed is the make-up of these components. Painting materials have gone from simple primitive stains, to twelfth century techniques such as fresco (applying stains to wet plaster), to tempera paints made from egg-yolk and powdered pigments, to oils, which were in use by the 1430's in the Netherlands by Jan van Eyck and others. Finally in the 1960's came the development of acrylic paints (Honour & Fleming, 1995).

Among the many traditional painting media used by artists today, none is given more deference and respect than oil. Oil paints were developed and rose to prominence in the 15th or 16th century, and this prominence has continued to the present day. In *The Artists' Handbook*, Mayer summarizes the key attributes that give oils their lasting appeal in the eyes of artists and art aficionados. Mayer lists the following attributes of oil paint. Though the majority lose their relevance when applied to computer-based painting, the first three are worth examining more closely[3]. Features of oil paint include:

> 1. Its great flexibility and ease of manipulation, and the wide range of varied effects that can be produced.
> 2. The artist's freedom to combine transparent and opaque effects, glaze and body color, in full range in the same painting.
> 3. The dispatch with which a number of effects can be obtained by a direct, simple technique.
> 4. The fact that the colors do not change to any great extent on drying; the color the artist puts down is, with very slight variation, the color he or she wants.
> 5. The fact that large pictures may be done on lightweight, easily transportable linen canvases.
> 6. The universal acceptance of oil painting by artists and the public, which has resulted in a universal availability of supplies, highly refined, developed, and standardized.
> 7. Its principal defects are the eventual darkening or yellowing of the oil, and the possible disintegration of the paint film by cracking, flaking off, etc. ((Mayer, 1991))

Issues of permanence, transportability, and availability of materials (items 4–6) are all made largely irrelevant by digital image manipulation tools. The single detriment listed, discoloration and cracking from aging (item 7), does not apply to digital creations. However, items 1–3 are all relevant to digital tools. In particular, item 3 is an area where computers tend to have difficulty: the "dispatch with which a number of effects can be obtained by a direct, simple technique." A strong case can be made that traditional painting tools and materials have been unmatched by modern computer programs for image generation in this regard.

[3] I have rearranged a few of these points so that those relevant to computer painting are all grouped together. The numbering is also mine.

Typical computer interfaces for painting software rely on a large number of states and modes to achieve their wide variety of effects. These modes generally become more difficult and unintuitive to manage the more flexibility the program allows. In contrast, the traditional tools of painting are in some ways very straightforward and intuitive to use, while still maintaining a phenomenal degree of flexibility. For example, one can obtain different marks by using different brushes or vary marks dynamically by applying different amounts of pressure and brush angle; one can make paint thinner with more oil medium for smoother strokes, or transparent glazes. Achieving these effects is second nature to a painter; the brush and paint become extensions of the artist.

Informal interviews I have conducted with many painters and digital artists over the course of this research have revealed that most of them crave this close connection with their materials, and feel it lacking in current digital tools. Even at a digital studio like Pixar Animation Studios, background painters will still create the detailed backgrounds that appear 3D animated films using traditional painting techniques. These paintings are then scanned into the computer for inclusion into the 3D virtual set.

One painter at Pixar sums up the situation like this (Tia, 2001):

> *"My artist's toolbox has oils and acrylics—and a computer. But the computer's not tactile, and I miss that. You can't push junk around. . . . [W]hen my computer goes down, I have no clue what to do. That's frustrating. But I really don't want to understand it"* — Tia.

This quote illustrates two distinct points. One is that most artists have absolutely no interest in becoming computer experts in order to use digital tools. The painters at a studio like Pixar are hired for their artistic talent, not their computer knowledge. The studio, in fact, spends at least three months training newly hired artists on the use of the in-house digital tools. This training is required in part because the digital tools work so differently from the traditional tools these artists are familiar with. This leads to the second point I would like to highlight: not only are current computer based painting programs significantly different, they also do not allow the same level of fluid, direct interaction with the medium that is possible with real painting, and desired by most painters. Digital paint models in typical programs do not allow the artist to "push junk around," nor are they "tactile".

This dissertation presents several novel techniques developed for simulating a natural medium like oil paint, as well as methods for simulating the brushes, canvas, and palette used in traditional painting. Together, the composition of these physically-based digital paint media and tools leads to a natural painting environment, which allows painters to work on the computer just as they would with real-world tools. When given a chance to use the more natural, 3D virtual simulation-based painting interface developed in the course of this research, painters have been very positive. Often the first question asked is how they can obtain a copy of the software for themselves.

My interactions with artists over the course of this research lead me to believe that, with the type of painting simulation and natural painting software interface presented in this dissertation, the divide between traditional artists and digital art can be significantly narrowed.

1.3 Applications and Benefits

By simulating the traditional tools and materials of painting, the potential advantages are several. First, as already mentioned, the primary goal is to create an effective and expressive tool for artists. By providing virtual equivalents of familiar tools, such a program can enable traditionally trained artists to more easily and rapidly make the transition to the digital realm. By transferring the skills and techniques they already know over to a computer-based system, they can immediately take advantage of the benefits of digital content creation, such as undo, save, no waiting for physical or chemical processes like drying, and unlimited version control. But even for artists who are not already trained painters, the techniques presented in this dissertation give artists more expressive virtual tools compared to existing painting programs.

The second application is as a tool for training novices to paint. Someone interested in learning painting techniques could use a simulation-based painting system to practice mixing paints to achieve the desired hue and tint, and to practice brush loading, basic strokes, layering and composition. This practice can be spread over multiple sessions separated by any amount of time, and the student can have complete control over the drying process, drying the canvas instantly so she can proceed immediately with no lost time, or leaving it wet indefinitely so she can continue working with a wet-on-wet technique after any length of hiatus. Furthermore, tedious setup and cleanup are eliminated entirely.

The third application for this technology is to increase the accessibility of painting. The potential exists to enable painters of all sorts to paint in situations in which they otherwise would not be able. The thinners used in oil painting, for instance, require a well-ventilated room for safe usage. This makes it difficult for anyone with limited space to pursue their painting hobby. Using a computer-based painting system produces no fumes, requires no physical space to store all the partially completed canvases, and creates no mess. It is perfect for children, who love to paint[4], but may have difficulty with the complex interfaces of most computer painting programs. Or it could be used effectively in a museum setting to get visitors more involved in thinking about the craft and skills that go into painting, as well as the science of visual thinking.

It is legitimate to ask why one should use the computer to simulate things the way they are, rather than how they could be. These arguments are raised cogently

[4]"Every child is an artist. The problem is how to remain an artist once he grows up."—Pablo Picasso

in (Schofield, 1994). Why not just ignore the constraints of the real world and create a new media truly unique to the computer? Simulated paint, no matter how faithful the reproduction, can never be more paint-like than paint itself, by definition. So why make the effort?

First, even in this dissertation, the goal is not to reproduce paint *exactly* as it is in reality. I allow for instantaneous drying, and for multiple levels of undo and redo, for instance, which are not easily achieved in the physical world. The philosophy is to model reality selectively, picking and choosing only aspects of the real world that are conducive to creative expression.

Second, a major tenet of this dissertation is that by creating a physically-based interface one leverages human beings' experience with how materials behave. So even if humans could adjust to some hypothetical non-physical rules, the user would have to take time to learn those unfamiliar rules. To the extent that the virtual materials mimic their real-world equivalents, no computer-specific training will be required to use them, thus resulting in a savings of time, money, and frustration.

Non-physical, or quasi-physical, paint systems that seek to transcend the real world are, nonetheless, a fascinating and important topic for further study.

1.4 Thesis

My thesis is:

> *With efficient, physically-based models for capturing the behavior, appearance, and feel of the brushes and thick paint used in traditional painting, one can create interactive, computer-based painting systems that are flexible, versatile, and easy to use.*

In support of this thesis I have created several prototype painting systems, dAb, Stokes Paint, and IMPaSTo, and have given them to hundreds of users to experiment with

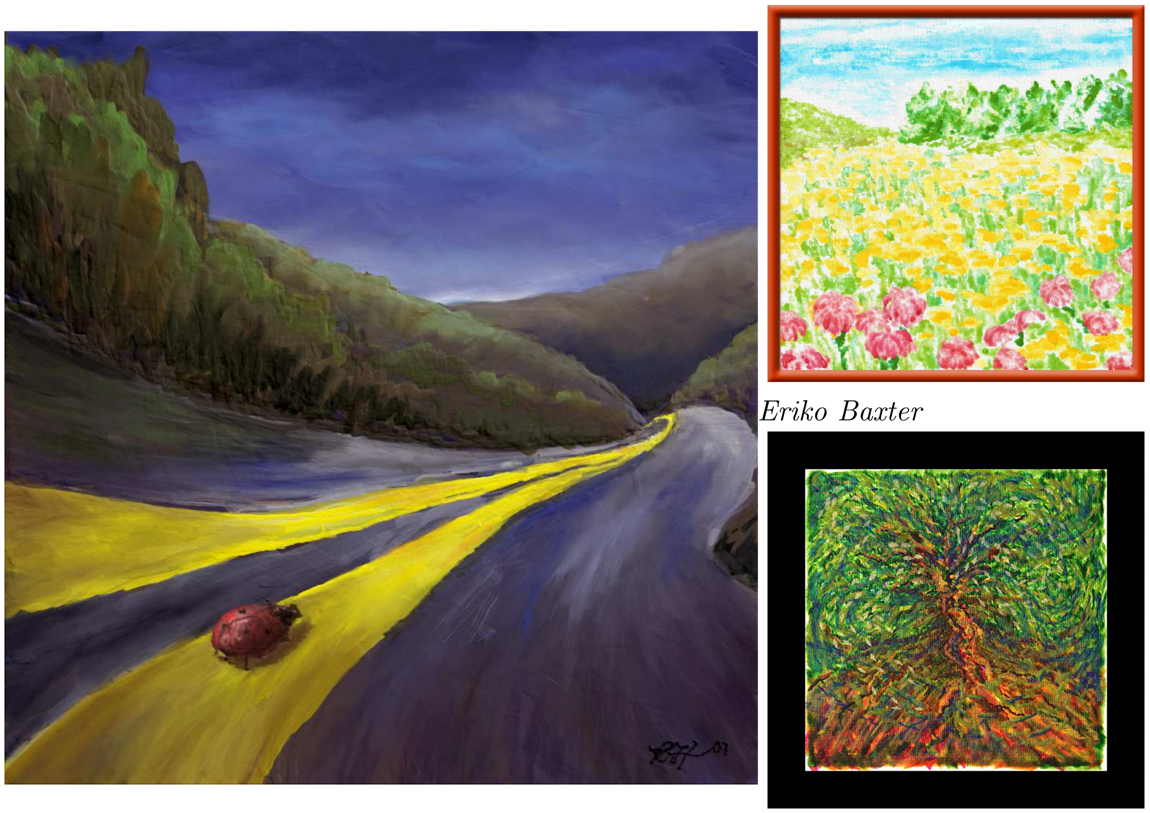


Figure 1.6: *Examples of paintings created by various artists using the different physically-based digital painting techniques presented in this dissertation. Many more examples are included in later chapters.*

and to create original paintings. The users range over all levels of experience, both with traditional painting techniques and with digital techniques, some being advanced at one or the other, some at both, and some complete novices at fine art. Regardless of experience, users have all been able to begin painting immediately, and many users have spent many hours enjoying the realistic painting experience, creating a wide variety of paintings. Examples of these paintings are shown in Figure 1.6. Many more examples are shown throughout the remaining chapters of this dissertation. Painters' reactions have generally been very positive; one went so far as to emphatically declare "this is *exactly* what I've been waiting for."

1.5 New Results

The goal of this dissertation is not simply to develop another digital image creation program, but rather to devise algorithms for creating a full simulation of traditional painting, using modern physically-based numerical simulation techniques. The result, I believe, is a simulation of painting that is not only unprecedented in its realism and in the versatility of the digital medium, but also unparalleled in usability. Though there are still many challenges to address in future research, this dissertation represents several significant strides towards the goal of creating a computer-based painting system that can compare to real-world painting in every way.

1.5.1 Overview of Results

I present several specific scientific advances to further the goal of realistic interactive digital painting. To adequately facilitate the painting process, one needs accurate models for both the brush and the paint, and this translates naturally into an overall painting pipeline consisting of the following simulation stages (Figure 1.7):

1. Brush dynamics: Update brush according to user input.

2. Paint dynamics: Update paint distribution according to brush motion.

3. Paint Rendering: Compute color, and display the resulting paint to the screen.

Each step is performed repeatedly until the artist deems the painting complete. The artist closes the loop by observing the results of each stroke and planning the next one accordingly in order to achieve a desired result. This feedback loop implies that a high level of interactivity is desirable in these models, in order to allow the user/artist to immediately see and react to the results of each manipulation. Consequently, the

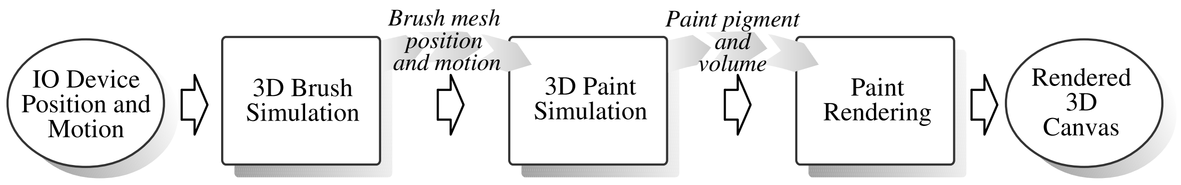


Figure 1.7: *System architecture overview. The process begins with input from either a pressure- and tilt-sensitive tablet or Phantom haptic IO device, then the system simulates the brush, simulates the paint, and renders the painting's pigment concentrations into a final RGB image for display.*

focus of this work is on techniques that are fully interactive, ideally providing 30-60Hz update rates.

The entire simulation should also be presented to the user via an easy-to-use interface. And since the feel of the materials is important to the overall experience of realism, an important aspect of the interface is the recreation of the haptic (or tactile) sensation of painting.

In this dissertation I describe novel models for each of the above components in order to create a painting system that is significantly closer to real-world, traditional painting than any previous computer-based system. I have combined these realistic, physically-based models with the first ever haptic interface for painting and a virtual mixing palette. Specifically, in the chapters that follow I present the following. 1) A model for 3D dynamically deformable paint brushes. 2) Three different models for paint behavior that fall in three distinct zones on the trade-off curve between performance and simulation fidelity. 3) An accurate real-time physically-based model for calculating the color of paint mixtures based on full-spectrum measurements of real oil paints. Finally, I present 4) a natural user-interface with haptic feedback models that truly give the painter the feel of painting.

The next few sections explain the responsibilities and scope of each component in the painting pipeline in more detail, and give a brief overview of my approach for each one.

1.5.2 Brush Dynamics

The brush dynamics component takes input from the user and transforms it into brush motion and deformation, and outputs a description of the brush's dynamic state. In many previous systems this transformation has been as simple as conversion from 2D mouse input to the 2D position of a brush bitmap on the canvas. In this dissertation it consists of a simulation of a deformable 3D brush driven by input from a six degree-of-freedom haptic interface. My simulation is based on quasi-static energy optimization of a skeletal brush representation, and includes an anisotropic friction model, and modeling of brush plasticity. The geometric model for the brush consists of a combination of a dynamic subdivision surface and individual bristles, both of which are deformed according to the skeletal dynamics.

Modeling of brushes is made difficult by the stiffness of most bristle materials, which leads to numerical stiffness in the differential equations used to model their motion(Witkin & Baraff, 1997). The modeling is further complicated by the intricate geometry of the thousands of individual bristles.

In Chapter 3, I present a hybrid approach to dealing with this complexity.

- Physical behavior and geometric modeling of the brush are treated separately.

- Physical behavior is modeled using a simplified spring skeletal system to capture the essential dynamics of a three dimensional brush under user control.

- The geometric model consists of two components, a surface model to capture the extents of the brush as a whole, and a bristle model to capture individual bristle details.

- The surface model consists of a subdivision surface that is dynamically deformed based on the deformation of the physical skeleton, and is able to mimic the paint marks created by smoother, softer brushes.

- The bristle model consists of hundreds of individual polygonal strips that are dynamically deformed using a weighted interpolation scheme based on the motion of the physical skeleton, capturing the scratchy, random look of strokes created by coarser brushes.

The flexible brush model I present offers the digital painter for the first time a range of virtual 3D deformable brushes that faithfully recreates the range of brushes that are typically found in the traditional painter's tool set. Combined with the paint models I will present, these brush modeling techniques are able to create a wide variety of realistic brush marks at interactive rates.

1.5.3 Paint Dynamics

The paint dynamics model takes the brush state description as input and determines how paint is altered on the canvas based on the brush's motion. In this dissertation I model the behavior of a viscous oil-like paint medium.

The complex geometry of real viscous paints that results from the simplest manipulation poses a challenge for real-time simulation. An example of this complexity is shown in Figure 1.8. The image shown was the result of working a dab of slightly thinned oil paint with a palette knife. Determining a good geometric representation is only part of the challenge. The motion and evolution of this surface when manipulated by a brush is even more difficult.

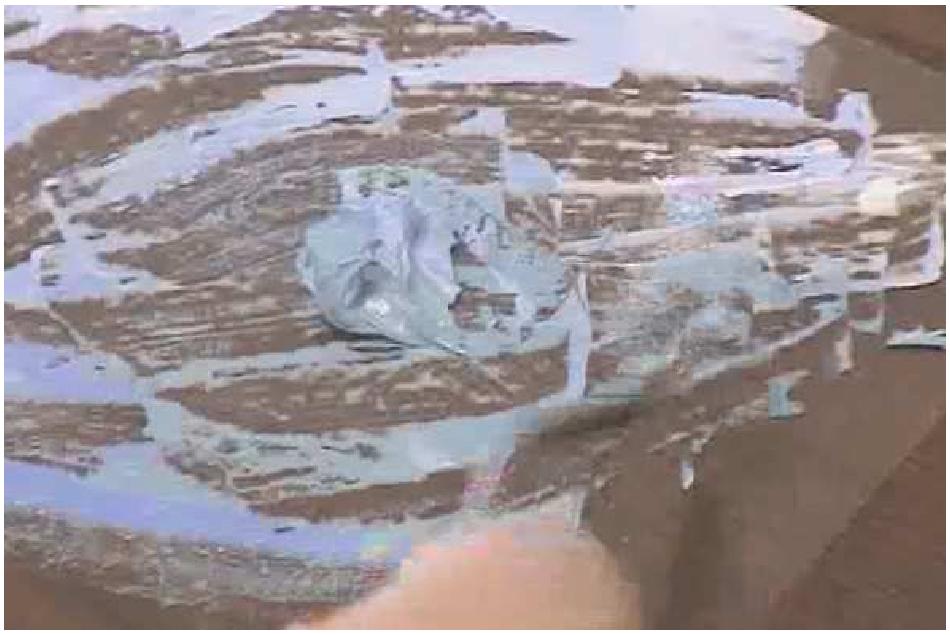


Figure 1.8: *The complex, dynamic geometric surface of real paint presents a challenge for simulation.*

Rather than a single approach, I introduce several methods for dealing with the complexity and the computational demands of manipulating a viscous fluid with a free surface and discuss the characteristics of each, for use on different computing platforms with different capabilities.

In particular, I present the following paint models:

- A simple 2D model appropriate for computers with a 500 MHz processor or better, and most any accelerated 3D graphics adapter. (Chapter 4)
- A method based on 3D viscous fluid flow equations for faster CPUs. (Chapter 5)
- A 2.5D method, based on physical principles, which leverages the parallelism and vector processing capability of programmable graphics processing units (GPUs). (Chapter 6)

Each paint model offers a particular trade-off between speed and physical fidelity. Figure 1.9 gives an overview of the major differences between these paint models.

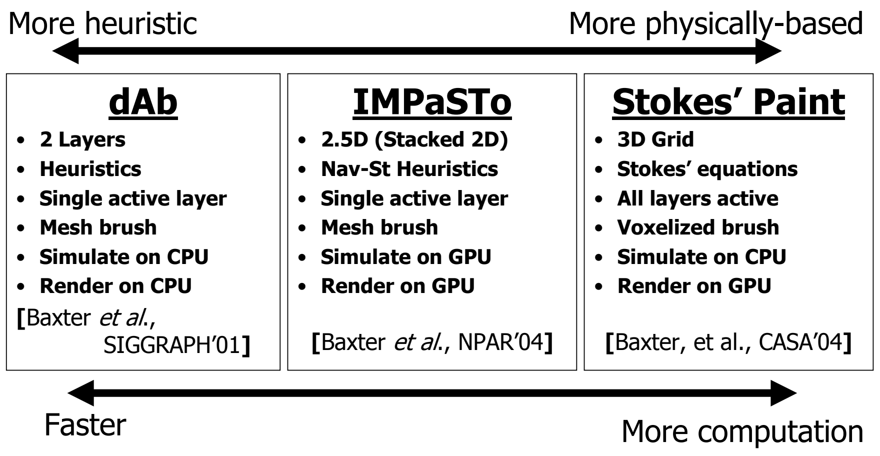


Figure 1.9: *A comparison of the three paint models I propose. In any type of simulation there is inevitably a trade-off between speed and simulation accuracy. The three paint models in this dissertation each occupy a different position along this trade-off curve.*

The first method, the subject of Chapter 4, is shown on the left in Figure 1.9. It is a simple two-layer, 2D heuristic that allows for fast interactive response even on modest hardware, but that still offers several paint-like attributes, and full integration with my 3D virtual brush models. This paint model was originally presented in (Baxter et al., 2001).

The second method, the subject of Chapter 5, is shown on the right in Figure 1.9. It is based on the Stokes' equations for viscous flow. It uses a full 3D grid to represent the paint and can therefore capture the interesting subsurface paint distribution that can arise even from fairly simple paint manipulation. It is also the most computationally demanding of my paint models. This paint model was originally presented in (Baxter et al., 2004a).

The third method, the subject of Chapter 6, is shown in the middle of Figure 1.9. It offers a balance between costly physically-based techniques and fast approximations. It uses a multiple-layer model to allow the user to build up an unlimited number of paint layers, and uses the programmable graphics processor to perform the simulation, leaving the CPU free for other tasks. This paint model was originally presented in (Baxter et al., 2004b).

1.5.4 Paint Appearance

Finally, given a distribution of paint on the canvas, the paint appearance model is responsible for rendering an image of that paint to the display.

Paint is made by mixing a dry pigment, such as titanium oxide powder, in a medium such as linseed oil. The optical properties of such a material are non-linear and cannot be modeled well using the simple, linear RGB color space that is typical in rendering computer graphics. The mixing of pigmented materials is much better modeled by the Kubelka-Munk mixing and compositing equations developed by German scientists Kubelka and Munk (Kubelka & Munk, 1931; Kubelka, 1948; Kubelka, 1954). These equations account for the internal absorption and scattering of light that occurs in a material such as paint.

As mentioned by Mayer (Mayer, 1991), the ability to create both transparent glazes and opaque effects is an important part of the flexibility of oil paint. To enable this range of behavior I present a multi-layer representation for paintings as part of the paint model.

Specifically, in Chapter 7 I present:

- A novel high-accuracy paint coloration model based on Kubelka and Munk's model, suitable both for interactive and archival color purposes.

- A Gaussian-quadrature technique for choosing an 8-sample color space basis at runtime.

- Techniques to enable relighting paintings under any full spectrum illuminant.

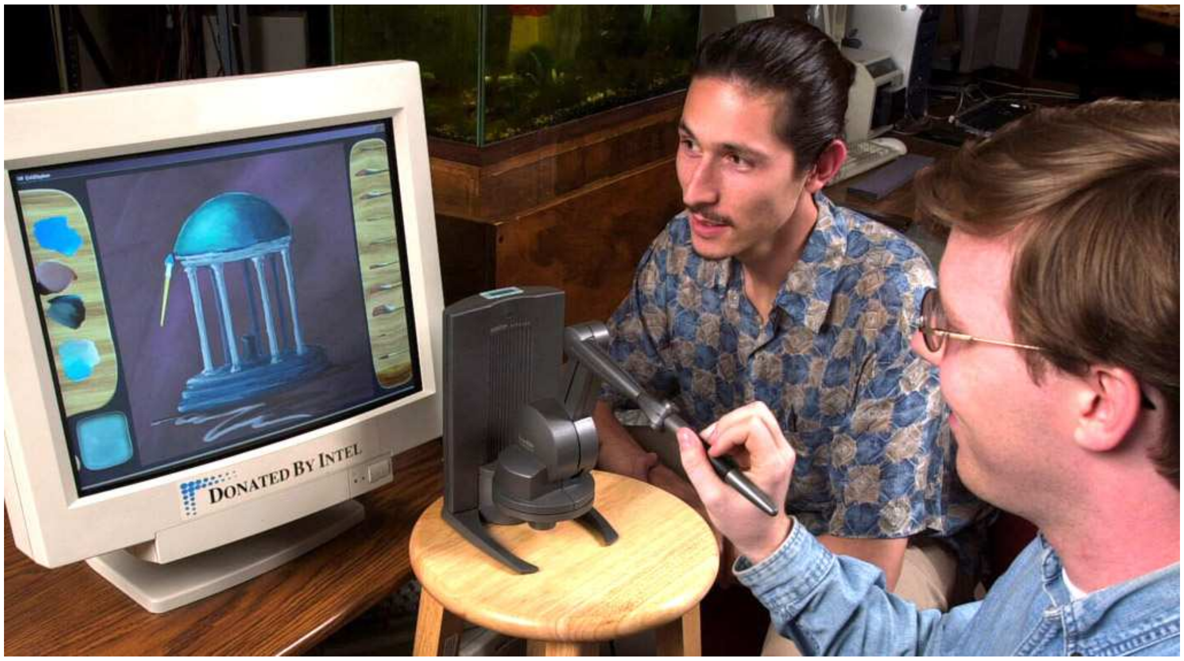


Figure 1.10: *Artists paint by manipulating an input device, like the haptic stylus shown above, to control a 3D virtual paint brush. The results are displayed interactively and continuously as the artist works.*

- A measurement technique for acquiring Kubelka-Munk coefficients from real-world paint samples.
- An implementation of the Kubelka-Munk color model that leverages the data parallelism of the GPU to deliver interactive performance.

1.5.5 Interface

An interface should be easy to learn but at the same time not limit the user. An interface that is intuitive, or, in other words, simply does "what the user expects" with little or no training, is highly desirable. In many cases a user's past experience determines the degree to which a new interface will seem intuitive. Keeping an interface simple by limiting the choices presented to the user can reduce the learning curve, but at the same time can restrict the flexibility afforded the user.

One striking benefit of using simulation extensively in a user interface for painting is that it enables a natural style of interaction that is not possible with the existing computer image generation tools. My interviews with users indicate that the interface introduced in this dissertation is easier to learn and less intimidating to many traditional artists than the interfaces typically used in image creation applications. At the same time, the natural 3D brush input system and simulated media gives the artist much of the same flexibility he or she would have with a real brush and paint.

The interface of a painting system should also serve to enhance the painter's connection with the virtual tools and materials. I propose a natural painting interface with haptic feedback that gives the user a fully three-dimensional input channel for interacting with the virtual brush and paint, with haptic feedback that gives the painter extra tactile information to better control the brush.

Specifically, in Chapter 8 I present:

- A natural interface for painting that recreates the traditional tools and environment of real-world painting.
- A painter's palette interface for brush loading and mixing that is intuitive and easy to use.
- Haptic models for brush force feedback. A simple model as well as a physically-based model for haptic interaction with fluids.

Figure 1.10 shows the overall interface setup.

1.5.6 Summary

This dissertation covers a wide range of topics important to the accurate, realistic simulation of the tools and materials used in traditional painting. Figure 1.6 gives a preview of some of the resulting paintings that have been created with the techniques in this dissertation, and Figure 1.10 shows painters using the natural interface. The techniques I present offer the painter an interface focused on process rather than product, with complex material models that facilitate expressive mark-making and serendipitous accidents.

1.6 Thesis Organization

The rest of the dissertation is organized as follows. The next chapter discusses previous work in painting systems and related subject areas. Chapter 3 details my novel physically-based virtual 3D brush model. Chapters 4–6 describe the three different models for paint dynamics that I have developed. In Chapter 7, I describe a technique for real-time rendering of the appearance of paint using the Kubelka-Munk equations. Chapter 8 details the issues relating to both the overall interface, as well as the specifics of haptic feedback generation used by this system. Finally, in Chapter 9, I summarize the major results of this dissertation and conclude with a look at areas for future improvement.

Chapter 2

Previous Work

Art is a process. [...] The computer also represents a process. But it is a polymorph of mathematical and logical design. What it can do is subject to what we believe it can do for us.
— (Schwartz, 1992)

In this chapter I describe the previous research in painting systems and non-photorealistic rendering that provides the general background and context for my work. The previous work related to the specific techniques I use in simulating brushes and paints based on physical models will be discussed later, in the corresponding chapters.

2.1 Non-Photorealistic Rendering Overview

The quest for photorealism has been the main driving force behind computer graphics research for the past several decades. Many stunning achievements have been made in this time. But a complementary research trend has gained momentum recently, one with the goal of creating more stylized imagery. Much as painters of the 19th century began to realize that realism was not always the best way to convey a message, recently graphics researchers have begun to embrace the idea that a simple line drawing can sometimes be a better way of conveying information than a realistically shaded 3D model, and that traditional artistic styles can be far more expressive than photorealistic images for some applications. The efforts to bring such stylization to computer graphics are collectively referred to as non-photorealistic Rendering (NPR). Research on NPR can be categorized roughly into two pursuits (Sousa, 1999): 1) rendering methods applied to directly to reference images and/or 3D models to create painterly or otherwise expressive renditions; and 2) simulation of natural media for drawing or painting (pencil, charcoal, pen-and-ink, watercolor, oil), including their correspondent tools and accessories (paper, brushes, canvas).

This dissertation is concerned with the latter, simulated media, and in particular the simulation of viscous oil-like paint and the attending tools: brushes, canvases, and palettes. Next I provide a survey some of the previous work in non-photorealistic rendering. More in-depth surveys and taxonomies can found in (Sousa, 1999) or in the books (Gooch & Gooch, 2001; Strothotte & Schlechtweg, 2002).

2.2 Automatic Rendering Techniques

The goal of automatic NPR methods is to algorithmically generate expressive renditions of existing photographs or 3D scenes, typically in some traditional artistic style. The challenge is to somehow distill the often fuzzy principles developed by artists over the years into concrete algorithms that can be executed by a computer with little or no human intervention.

Various methods for automatically rendering images in pencil sketch, charcoal, pen-and-ink, crosshatch, technical illustration, cartoon shading and other styles have been developed. Chapter 2 of (Sousa, 1999) gives a good summary of the work done in these areas throughout the 1990s. Two books have been published on the subject of NPR as well (Gooch & Gooch, 2001; Strothotte & Schlechtweg, 2002). Both offer excellent summaries of the state of the field.

More relevant to this dissertation, several researchers have also developed automatic methods for transforming images or 3D scenes into *painterly* renderings (i.e. renderings that look like paintings). Some examples include (Meier, 1996; Curtis et al., 1997; Litwinowicz, 1997; Hertzmann, 1998; Hertzmann, 2001; Hertzmann, 2002; Hays & Essa, 2004). These algorithms place and align paint strokes based on properties of the input image, such as image silhouettes or isochromal contours. Most of the work in this area has treated strokes merely in terms of color, but the latter two references additionally use height fields and bump-mapping to generate strokes with surface texture as well.

While most techniques so far have focused on either one or the other of the main NPR pursuits – either on automatic stylization or on providing a model for natural media – the watercolor simulation of (Curtis et al., 1997) was notable in that it achieved both. It presents first a realistic simulation of the medium, then uses that simulation as a basis for a mostly automatic NPR technique. (Sousa, 1999) accomplished this as well with his pencil modeling and subsequent automatic pencil rendering algorithms.

Automatic stylized depiction is an important goal; however, in many environments it is more important that artists have total control over the resulting work, which runs counter to the goal of having the computer algorithmically determine where strokes will be placed. The best way to give the artist full control is to provide him or her with an interactive model of the media. Given such a realistic media simulation, existing techniques can be used to automatically guide the virtual brush and place strokes to generate a stylistic rendering automatically, as demonstrated by the work of (Curtis et al., 1997) and (Sousa, 1999).

I therefore concentrate in this dissertation on the problem of how to simulate a viscous paint medium and paint brushes realistically in real time, and in how to provide the artist with an efficient human-computer interface that enhances the interactive painting experience.

2.3 Modeling Natural Media

Many models for simulating natural media have been developed over the years. Table 2.1 presents a summary of the features of many previous systems that specifically modeled paint. I have included several of the early 2D painting programs that really provided more of a “bitmap” style of painting than an approximation of any real medium; nevertheless, their creators were clearly attempting to simulate the look and feel of painting as nearly as they could, given the hardware available at the time. The evidence is in the nomenclature they used. For example, Richard Shoup’s 1973 “SuperPaint” program (Shoup, 2001)[1] and “Paint” by Alvy Ray Smith (Smith, 1978), both provided the user with “brushes”, a “palette”, “canvas” and “paint”. The intent to create something like real-world painting is clear, even though the “brushes” were little more than 2D bitmaps stamped onto the framebuffer, and the “palette” was simply a list of colors. (Smith, 2001) presents an entertaining history of other various early painting systems and the individuals behind them.

(Lewis, 1984) was one of the first to suggest a method for giving strokes created in painting programs more of the organic character of natural media. He chose not to model a particular medium, but rather to synthesize textures that have spectral characteristics similar to natural materials. The result was images that were significantly more organic in appearance than the typical images of the day.

(Strassmann, 1986) provided one of the earliest models for a specific natural medium. His “hairy brushes” for ink painting modeled the the amount of ink contained in each bristle over the course of a stroke, and allowed a brush to deform over the course of a stroke, as well. The input to the system was cumbersome, however, requiring the user to specify spline control points, and features like paper texture or the mixing of ink through diffusion were not modeled.

[1]The SuperPaint project was initiated in 1972 and completed in 1973.

Painting System	Media type	Brush	Brush dynamics	Haptics	Grain	3D Canvas	Palette	Color	Input	Input DoF	Notes
(Shoup, 2001)	Bitmap	2D	None	No	No	No	Simple	RGB	Light pen	2	
(Smith, 1978)	Bitmap	2D	None	No	No	No	Simple	RGB	Light pen	2	
(Whitted, 1983)	Bitmap	2D	None	No	No	No	N/A	RGB	-	2	
(Lewis, 1984)	Bitmap	2D	None	No	Yes	No	None	RGB	Mouse	2	
(Greene, 1985)	Bitmap	3D	Real	Static	No	No	None	RGB	Real brush	6+	
(Strassmann, 1986)	Ink	1D	Splines	No	No	No	None	Mono	Mouse	2	
(Bleser et al., 1988)	Charcoal	2D	Bitmap set	No	Yes	No	None	Mono	Tablet	5	
(Posch & Fellner, 1989)	Bitmap	2D	None	No	No	No	None	RGB	-	2	
(Ware & Baxter, 1989)	Bitmap	2D	Size, Color	No	No	No	?	RGB	Tracker	6	
(Williams, 1990)	Z/Bitmap	2D	None	No	No	No	Simple	RGB	Mouse	2	
(Hanrahan & Haeberli, 1990)	Bitmap	2D	None	No	No	Yes	Simple	RGB	Mouse	2	
(Pham, 1991)	Ink	2D	None	No	No	No	None	Mono	Mouse	2	
(Guo & Kunii, 1991)	Ink	2D	None	No	No	No	None	Mono	Mouse	2	
(Small, 1991)	Watercolor	2D	None	No	No	No	None	Mono	Mouse	2	
(Cockshott, 1991; Cockshott et al., 1992)	Oil (thin)	2D	None	No	No	No	Simple	RGB	Mouse	2	
(Hsu et al., 1993; Hsu & Lee, 1994)	Multiple	2D	None	No	No	No	Simple	RGB	Mouse	2	
(Kunii et al., 1995)	Ink	2D	None	No	Yes	No	None	Mono	Mouse	2	
(Agrawala et al., 1995)	Bitmap	3D	None	Static	No	Yes	Simple	RGB	Tracker	6	
(Curtis et al., 1997)	Watercolor	2D	None	No	Yes	No	Simple	KM3	Mouse	2	
(Zhang et al., 1999)	Ink	2D	Size	No	Yes	No	None	Mono	Tablet	3	
(Lee, 1999; Lee, 2001)	Ink	3D	Offline	No	Yes	No	None	Mono	Tablet	3	
(Saito & Nakajima, 1999)	Ink	3D	Optimization	No	Yes	No	Simple	KM3	Tablet	3	
(Wong & Ip, 2000)	Ink	3D	Splines	No	No	No	None	Mono	Mouse	2	
(Yu et al., 2002; Yu et al., 2003)	Ink	2D	Size	No	Yes	No	None	Mono	Tablet	3	
(Chu & Tai, 2002; Chu & Tai, 2004)	Ink	3D	Optimization	Static	No	No	Simple	RGB	Tracker/Tablet	6/5	
(Yeh et al., 2002)	Ink	3D	Newton	Yes	No	No	Mixing	RGB	Phantom	6	
(Xu et al., 2002; Xu et al., 2003)	Ink	3D	Offline/Table	No	No	No	Simple	RGB	Mouse	2	
(Chan & Akleman, 2002)	Ink	2D	Size	No	No	No	None	RGB	Tablet	3	
(Guo & Kunii, 2003)	Ink	2D	Splines	No	Yes	No	None	Mono	Mouse	2	
(Huang et al., 2003)	Ink	2D	Splines	No	Yes	No	None	Mono	Mouse	2	
(Lin & Shih, 2004)	Ink	2D	Size	No	Yes	No	Simple	KM3	Tablet	3	
(Mi et al., 2004)	Ink	2D	Size	No	No	No	None	RGB	Tablet	3	
(Adams et al., 2004)	Multiple	3D	Newton	Yes	No	Yes	Mixing	RGB	Phantom	6	
(Laerhoven et al., 2004b)	Watercolor	2D	No	No	Yes	No	Simple	RGB	Mouse	2	
(Painter 8, 2003)	Multiple	2D	Bitmap set	No	Yes	No	Mixing	RGB	Tablet	5	CMY palette
(ArtRage, 2004)	Multiple	2D	Bitmap set?	No	Yes	No	Simple	RGB	Tablet	5	
(Expression 3, 2004)	Multiple	2D	Size	No	Yes	No	Simple	RGB	Tablet	3	
(Z-Brush, 2000)	Z/Bitmap	2D	?	No	Yes	Yes?	Simple?	RGB	Tablet	3	
(Deep Paint, 2000)	Multiple	2D	Bitmap set?	No	Yes	No	Simple?	RGB	Tablet	3	
This dissertation:											
(Baxter et al., 2001)	Oil	3D	Newton	Yes	No	No	Mixing	RGB	Phantom	6	
(Baxter et al., 2004a)	Oil	3D	Newton	Yes	Yes	No	Mixing	KM8	Phantom/Tablet	6/5	
(Baxter et al., 2004b)	Oil	3D	Newton	Yes	Yes	No	Mixing	KM8	Phantom/Tablet	6/5	
(Baxter & Lin, 2004)	Oil	3D	Optimization	Yes	No	No	Mixing	RGB	Phantom/Tablet	6/5	

Table 2.1: *Feature comparison for a selection of painting systems.*

Since Strassmann's "Hairy Brushes", the ink used in traditional Chinese or Japanese calligraphy has become a popular subject of investigation in NPR. The pictures in this style, in Japanese called either *sumi-e* or *suiboku-ga*, are characterized by a few deftly-placed strokes to convey the subject completely. The ink is very thin and tends to diffuse into the paper in a complex way that depends upon the paper grain. Researchers modeling oriental ink try to capture these effects. (Strassmann, 1986; Guo & Kunii, 1991; Kunii et al., 1995; Zhang et al., 1999; Lee, 1999; Lee, 2001; Saito & Nakajima, 1999; Saito & Nakajima, 2000; Wong & Ip, 2000; Yu et al., 2002; Yu et al., 2003; Chu & Tai, 2002; Chu & Tai, 2004; Yeh et al., 2002; Xu et al., 2002; Xu et al., 2003). Many of these references perform some sort of diffusion calculation to determine the spread of the ink in the paper. They use either a cellular automata model like (Zhang et al., 1999) or diffusion models based on partial differential equations (Xu et al., 2003). A common approach for dealing with paper grain and its effect on diffusion, introduced first in (Guo & Kunii, 1991), is to model the paper by laying virtual fibers down onto the surface and then determining diffusion coefficients between adjacent cells based on the number of fibers that connect them.

Tunde Cockshott, a painter turned computer scientist, developed the "Wet & Sticky" model for paint (Cockshott, 1991; Cockshott et al., 1992), inspired by his own dissatisfaction with the "shallow view of the process of real painting" he saw in existing painting programs. His basic model is a cellular automaton that exchanges units of paint back and forth between adjacent canvas cells based on a set of rules and on the properties stored in "particles" of paint. The overall concept is similar to some of the ink models mentioned above, though the specific rules differ. His rules include transfer of paint from cells that are more full to cells that are less full, and increased exchange in the direction of gravity, and flow inhibition due to surface tension. The underlying computational complexity was high for real-time implementation on machines of the day, so a model for brushes or for interacting with the paint was never proposed.

The vision and goals of my dissertation are similar to those of (Cockshott, 1991). Nonetheless, our focus and approach differ significantly. He focuses on the paint itself and the "ambient behavior" of paints that run, drip, and diffuse on their own without direct human intervention, whereas my focus is firmly planted in the critical point of interaction between the human-controlled brush and the passive paint, and how that human interaction influences the paint locally. In this respect my work can be seen as complementary to that of Cockshott, who explicitly avoids questions of brush modeling and active brush-paint interaction, deferring these issues to future work. I focus precisely on those topics, and neglect the issue of modeling paint's ambient behavior. The two systems are complementary and could easily be used in conjunction with one another.

The watercolor model of (Curtis et al., 1997) used a simplified physical model, based in part on the shallow water equations, in order to recreate several very specific features seen in real watercolor paintings: back-runs, dry-brush effects, diffusion, and edge-darkening. Both (Curtis et al., 1997) and (Cockshott et al., 1992) describe a philosophy of modeling "just enough" of the physics to get plausible behavior with reasonable computational cost. To a large extent, that is the approach taken by this dissertation as well. Recently, (Laerhoven et al., 2004b; Laerhoven et al., 2004a) have begun to reimplement the work of Curtis *et al.* using a distributed parallel architecture to enable the computation to proceed in real-time. The authors have indicated that a version running on graphics processors (GPUs) is also under investigation.

Several recent commercial painting packages have also included natural media emulations. Corel's Painter (Painter 8, 2003; Zimmer, 1994) is able to achieve several types of realistic-looking natural media, including thick paint, by clever use of sets of

2D textures, bump-maps, and compositing tricks. The paint model looks paint-like; however, it lacks any real material model, and so the paint cannot be pushed around or worked on the canvas like real paint. ArtRage from Ambient Design (ArtRage, 2004) provides a thick paint model similar to that in Painter. Again, the paint looks fairly realistic but does not behave realistically, and cannot be manipulated like real paint. The Deep Paint program from Right Hemisphere (Deep Paint, 2000) appears to be the closest to providing a simulation of thick paint, by allowing all brush effects to optionally apply to a heightfield layer in addition to the color layer.

A related area is that of 3D paint systems, systems that can be used to paint onto three-dimensional surfaces. (Hanrahan & Haeberli, 1990) were the first to present a system to do this using 2D input and projectively mapping a 2D brush bitmap onto the 3D surface. The paper also presented the idea of painting various attributes other than color, such as glossiness (specularity) or bumpiness. (Agrawala et al., 1995) presented a method for painting a scanned surface by registering the scanned data in the computer with the real object, and then painting on the real object with a tracked 6-DOF stylus. The main problem reported was that one needed to divide attention between the real object and the image on the monitor in order to paint. (Bandyopadhyay et al., 2001) provided a solution to this problem by displaying the color directly on the real object object using digital light projectors. Recently (Adams et al., 2004) presented a 3D painting system for point-sampled models, with paint and brush models based on those in (Baxter et al., 2001). The key novel feature of this system is its ability to dynamically and adaptively resample the model to adequately represent any scale of paint stroke detail. In terms of commercial products, Deep Paint 3D (Deep Paint 3D, 2000) provides a more modern implementation of the projective painting techniques in (Hanrahan & Haeberli, 1990).

Using "3D Paint" to mean something completely different, (Williams, 1990) presented a system for *creating* 3D objects by painting with height, or depth instead of color. The commercial product ZBrush (Z-Brush, 2000) takes this idea further, allowing for both painting with depth to create surfaces as well as painting existing surfaces with attributes as in the 3D painting system of (Hanrahan & Haeberli, 1990).

Aside from paint media, many models for other natural media have also been presented. (Bleser et al., 1988) presented a model for charcoal, implemented as a matrix of bitmaps from which one is dynamically selected based on the pressure and tilt of the stylus. A similar bitmap selection technique is also described in (Zimmer, 1998). (Saito & Takahashi, 1990) presented the G-buffer, a framework for extracting various geometric information from a model for use in creating many sorts of "comprehensible renderings," such as silhouette drawings. The concepts they present were built upon by many of the subsequent NPR researchers, including (Winkenbach & Salesin, 1994; Winkenbach & Salesin, 1996), who present a model for pen-and-ink line drawings. Their pen-and-ink model dynamically performs crosshatching based on 3D lighting calculations and a 3D model's parameterization. Schofield's Piranesi system combines NPR rendering techniques with a strong component of interactive artistic control (Schofield, 1994), resulting in a new kind of computer-based NPR medium as opposed to a recreation or simulation of an existing medium. Sousa's dissertation (Sousa, 1999) presents a carefully constructed model for pencil drawing based on micrograph measurements of actual pencil marks on paper. (Sourin, 2001) presents a model for metal embossing and woodcutting that creates cuts and features by modifying a functional representation of a surface. In (Rudolf et al., 2003; Rudolf et al., 2004) a method is described for simulating wax crayons that accounts for the semi-translucency of real crayon as well as the wear on the crayon itself. (Wyvill et al., 2004) presents a technique for simulating the cracks that appear in *batik* paintings, an Indonesian art form.

Despite all of these models for paints and other media, prior to this dissertation there has not been a fully interactive physically-based simulation of a thick oil-like painting medium. As noted, many methods have been proposed for the thin ink style of paint used in Oriental calligraphy and *sumi-e* painting. The watercolor simulation of (Curtis et al., 1997) reproduced many effects observed in real watercolor quite closely, but watercolor and ink are both very different from oil paint. The “Wet & Sticky” paint presented by (Cockshott et al., 1992) incorporated diffusive and running effects, behaviors that are really more characteristic of thinner paints.

2.4 Painting Interfaces

The interface of a painting program is an important consideration insofar as it is the painter’s sole means of communicating intent to the program. An important part of the interface is the input device used. If you consider just the handle portion of a real brush, the part that the painter grasps, it has six degrees of freedom (DOF), as does any unconstrained rigid body. By considering the bristles as well, one could count a practically unlimited number of freedoms in the brush; however, for characterizing the interface it is those degrees of freedom directly controlled by the user that matter. Painters use all six degrees of freedom to their maximum advantage to create a wide variety of strokes from a single brush.

Early paint programs, however, provided just 2-DOF input via a tablet or mouse, allowing control only over the X and Y position of the brush on the screen (See Table 2.1). It could be said that interacting with the computer through a 2-DOF mouse is like to trying see the world through a soda straw. It is a very narrow channel for communication when compared with the full capabilities of human hands. A 2-

DOF input device for painting severely limits the expressive potential and flexibility of a virtual brush compared to that of a real brush.

Later tablet interfaces added a third DOF, a pressure sensor. In these systems the pressure typically controls the size of the brush footprint, allowing the painter to create strokes with more variability. Other tablets, including the most popular commercial tablets available today, have added X and Y tilt sensitivity also for a total of 5-DOF input (X,Y,pressure,X-tilt,Y-tilt). Compared with a 6-DOF interface these tablets lack only the ability to twist the brush about its axis, and thus are fairly capable input devices for painting. Still, current painting programs take little advantage of these extra degrees of freedom.

Another class of input devices that have been used are spatial trackers. These are 6-DOF input devices tracked in space using a variety of different technologies: ultrasound, magnetic or electro-magnetic fields, or optical sensors. The chief problem with using a tracker as a painting input device is that there is no force feedback. (Chu & Tai, 2002) solve this problem by attaching tracking devices to a real brush and then calibrating the position of the virtual canvas with a physical surface, but this has some drawbacks as well. The main problem is that the actual brush may not be deforming in the same way as the simulated virtual brush, and so the force feedback and visual representation shown to the user can be completely out of synchronization.

Finally there are 6-DOF interfaces like the Phantom (Massie & Salisbury, 1994), which in addition to 6-DOF input also offer program-controlled haptic (force feedback) output (See Figure 2.1). With these input devices the program can ensure that the haptic feedback presented to the user is consistent with the simulation state in the computer. It also allows the brush characteristics such as stiffness to be modified on the fly.

In the history of input devices for painting there is one that appears to have had

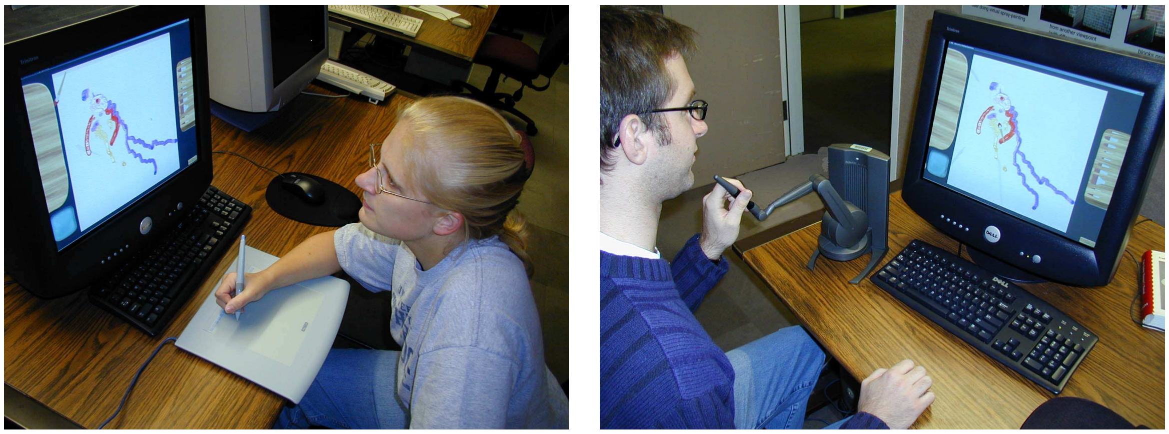


Figure 2.1: *In my system, user input is obtained either through a 5-DOF Wacom™ tablet (left) or a 6-DOF Phantom™ haptic interface (right).*

little long-term influence, but that deserves special mention nonetheless. (Greene, 1985) introduced a unique interface for painting called the "Drawing Prism". This device used a specially designed prism and camera setup to capture the contact footprint of real objects with the surface of the prism in real time. In this way one could paint using any real brush (or finger, or any other object) and the actual contact area of the bristles on the surface would be used to determine where paint was deposited on the virtual canvas. In some ways this is the ideal painting interface; however, it could not allow for effects such as complex brush loading since it could only determine a monochrome contact mask. A modern implementation of the drawing prism using color cameras could possibly identify specific brush regions on a color coded brush thereby allow for complex loading.

Chapter 3

Brush Modeling

"There is no item of greater importance to the successful execution of a painting than a sufficient quantity of the very-highest-grade brushes that it is possible to find. It is one department of the artist's equipment where no skimping or compromise should be allowed; he may go without or use makeshift supplies of some items but poor brushes are a severe handicap to good painting."

It is with the above quote that Ralph Mayer introduces the subject of brushes in *The Artist's Handbook* (Mayer, 1991). Having the proper brushes is critical to good painting. A good set of brushes can enable a competent artist to quickly create virtually any effect he or she can imagine, from the intricate detail of cresting waves, leafy trees and delicate flower petals, to wispy billowing clouds, and the subtly blended shifting hues in a sunset. Digital artists can benefit greatly from having this expressive power available to them in computer painting programs.

In this chapter, I will describe vBRUSH, a model for three-dimensional, flexible virtual paint brushes, that provides painters with a realistic recreation of real-world paint brushes on a computer. This brush model was originally presented in (Baxter & Lin, 2004).

Though there are many types of brushes commonly used in painting (see Figure 3.1), they share certain properties that make them effective tools for applying paint to a surface in accordance with an artist's intentions. Artistic references such as (Mayer,

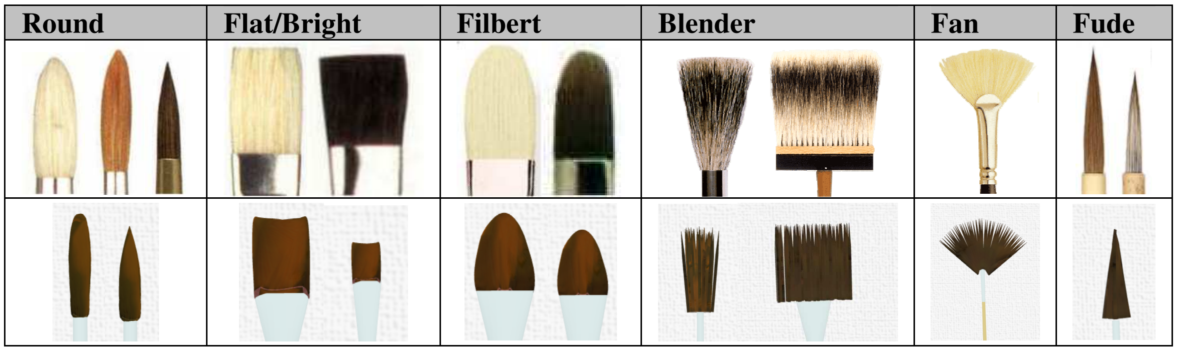


Figure 3.1: *A variety of real brushes and an example of a vBRUSH modeled version of each.*

1991) describe desirable attributes of high-quality brushes with terms such as "elasticity", "durability" and "ability to maintain a point". For a virtual brush, however, the greatest challenge is capturing the most basic physical attributes such as stability, passivity, interactivity, and proper reaction to friction, which are all taken for granted in the physical world.

Brushes are generally very stiff dynamical systems, due to a high stress-to-mass ratio, which leads to large accelerations, so it is difficult to simulate them with stability and accuracy using time-stepping integration techniques, especially when friction must be accurately accounted for (Witkin & Baraff, 1997).

The vBRUSH model simulates the dynamics of brushes using an optimization-based framework, similar to (Chu & Tai, 2002; Saito & Nakajima, 1999). However, vBRUSH incorporates a versatile brush construction methodology and introduces individual bristle dynamics to achieve detailed bristle effects. The underlying dynamic model for a brush is created from multiple optimization-based spine primitives. The geometry of the brush head is then created out of a combination of subdivision surfaces and thin polygonal strips. The deformation of the spines determines both the motion of the subdivision surfaces as well as the strips. With this approach, brushes composed of a subdivision surface and hundreds of strips can be simulated at interactive rates. The key characteristics of the vBRUSH brush model include:

- A complete set of high-quality virtual 3D brushes for the digital studio, including models for rounds, flats, brights, filberts, badger blenders, and a fan brush, capturing the behavior of both Oriental and Western-style brushes;
- A geometric representation empowering easy creation of fine bristle features and allowing for both smooth, clean strokes and rough, scratchy marks;
- A constrained dynamics framework capable of handling extreme deformations like bristle splaying and modeling anisotropic friction and brush plasticity;
- Versatile modeling tools for enabling non-programmers to produce almost any other shape or type of brush desired.

Several amateur artists have used the vBRUSH realistic virtual brush models to create paintings exhibiting complex brush marks and fine bristle detail as shown in Figures 3.11–3.13.

In order to give the artist full control over the virtual brush, I use input devices with at least 5-DOF input, preferably 6-DOF. The input devices used to control these brushes will be described in detail in Chapter 8.

The remainder of this chapter is organized as follows. I first review related work, then describe my dynamic model for brushes, and finally the geometric model. I conclude with demonstrations of the range of marks possible and paintings created with the brush model.

3.1 Previous work

The first tool for interactively creating graphical figures on a computer was Ivan Sutherland's visionary 1963 SketchPad system (Sutherland, 1963). Though revolutionary, this was a tool for creating technical illustrations and precise mechanical drawings, not paintings, so it did not include any notion of a brush. However, it opened up a whole new category of computer tools for image creation. According to Alvy Ray Smith (Smith, 1997; Smith, 2001), the very first crude color painting program was developed by Joan Miller at Bell Labs in 1969. It used a 3-bit color framebuffer. It is not clear from Smith's account when the concept of using a 2D "brush" to "paint" pixels in a frame buffer came about, but it seems to have existed from very earliest days of digital paiting. Certainly the concept of a brush was present in the subsequent systems by Shoup, Smith and others, and over time the idea was expanded to include features such as "z-paint" (using an additional depth value to modify marks), and various blending and image processing operations as well. Most painting programs today still use the same basic organization and conceptual model as Shoup, including widely used programs like Adobe Photoshop(Photoshop, 2004). Cockshott (Cockshott, 1991) refers to all these as "Shoup model" painting programs, since the majority of the core features of these programs were present in Shoup's 1973 SuperPaint system(Shoup, 2001).

Computer painting programs have matured significantly since their debut, but most still use this "Shoup model" and deposit paint on the surface using a two dimensional brush that "paints" an image buffer by repeatedly stamping a fixed 2D bitmap.

Several researchers have endeavored to more accurately model the *appearance* of real brush marks without developing a full 3D brush model by means of 2D heuristics. Strassmann modeled a brush as a one-dimensional array of bristles swept over a trajectory defined by a cubic spline curve (Strassmann, 1986). His work was able to account for several effects achievable with an actual brush, such as varying color, width, and wetness. In the area of automatic painterly rendering, (Hertzmann, 2002) and (Hays & Essa, 2004) have used 2D bump maps to represent individual brush marks, and created paintings automatically from images by overlapping many such marks. The bumpy texture of these marks, however, comes from a predetermined bump map, not from any dynamic physical model of brush bristles.

Wong and Ip (Wong & Ip, 2000) defined a complex set of interrelated parameters to vary the density, opacity, and shape of a brush footprint in a way that takes into account the behavior of a three-dimensional round calligraphy brush. The resulting stroke appearances are *guided* by the plausible behavior of a brush, but are not actually physically generated. The method as described is only partially interactive, and the user input required to control the appearance of each stroke seems tedious.

The approach for brush modeling I present bears some similarity to the work of Saito (Saito & Nakajima, 1999) on modeling a physical 3D brush for Japanese calligraphy and *sumie* paintings. However, the technique presented in this dissertation is more flexible in terms of brush shape, dynamics, and loading, and is able to take advantage of 3D graphics hardware as well.

The "Virtual Chinese Brush" of Chu and Tai (Chu & Tai, 2002) delivers a very convincing model for Chinese calligraphy that includes factors such as plasticity, tip spreading, and "pore resistance" (the tendency of the bristles to get stuck on the rough surface of the paper). Like Saito, Chu and Tai also used optimization for the brush dynamics, but with a more elaborate internal structural model with lateral springs. Their surface model is a swept surface of ellipses of diminishing radius. This is a very good model for round, oriental calligraphy brushes, but it is not able to model the wide variety of brush types used by western painters.

The Chinese calligraphy brush model in (Xu et al., 2003) represents the brush head as a collection of individual NURBS tufts, which can dynamically and recursively split into smaller NURBS tufts based on brush deformation. The dynamic model consists of heuristics for modifying surface control parameters based on brush position history. Realistic stroke results are achieved by a model-based technique in which a large set of model parameters are trained using several real example strokes in order to generate a particular type of mark.

Optimization has also been used in other areas of computer graphics for creating physically-based animation. One area is cloth simulation (House & Breen, 2000), mainly computing static drape. For cloth, these techniques have largely been superseded, because they do not handle inertial effects well, which are important in dynamic cloth. Motion retargeting and editing is another major area where optimization is used with much success, e.g. (Gleicher, 1998), although typically the problems are solved off-line. (Witkin & Kass, 1988) also used a space-time optimization technique to generate smooth physically-plausible character animations that interpolate key poses.

3.2 Introduction to Brushes

The brushes used in traditional painting take a wide variety of shapes and forms. Fig. 3.2 shows the anatomy of a typical brush, which consists of a handle, ferrule, and

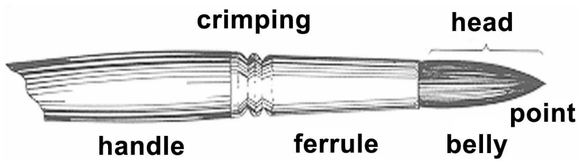


Figure 3.2: *Basic brush anatomy*

the head. The brush head is further subdivided into the belly and the point. Brush heads are made with a variety of bristles, natural soft animal hair, including red or black sable, badger hair, fitch or ox hair, hog or boar bristles, and synthetic materials.

Some of the most common styles for brushes used in oil-like painting (Mayer, 1991) are:

- **Rounds.** Have a simple tubular shape with a semi-blunt point, allowing for a great variety of strokes.
- **Flats.** Thinner and wider than rounds with bristles squared off at the point. Flats are typically longer than they are wide.
- **Brights.** The same shape and construction as flats but typically shorter, with width nearly equal to length.
- **Filberts.** Have a thicker collection of bristles that increase their ability to hold paint. Filberts usually have oval-shaped heads. The shape of their tips is formed by placing naturally curved bristles on the outer edges of the tip so that they curve inward, which helps counteract the tendency of the tip to splay outward with repeated use.
- **Badger Blenders.** Traditionally made of fine badger hair, round in shape, with the bristles flaring out at the end rather than tapering to a point. Used mostly for blending to create smooth color transitions.
- **Fans.** Sometimes used for delicate or wispy manipulations of wet paint, and also used for blending paint, like the badger blender.
- **Fude[1].** A round Japanese calligraphy brush typically made with longer hairs, able to form a good point for fine lines, or create wide or scratchy strokes when pushed harder or used on edge.

[1]Pronounced *foo-day.*

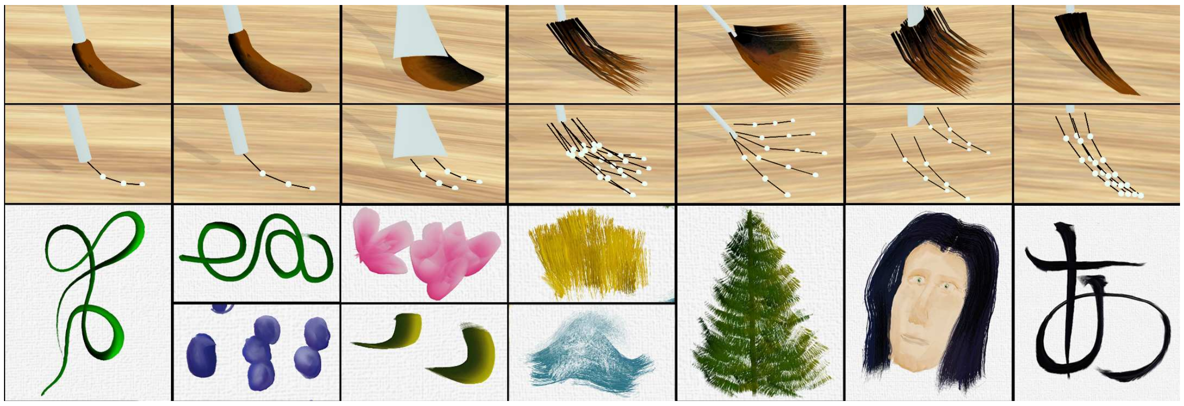


Figure 3.3: *Some vBRUSH brushes, their spine models, and example marks made with each.*

There are many other types of brushes, such as liners, chisels, mops etc. The vBRUSH approach can model any of these, the brushes above just represent some of the most versatile and widely used varieties. Figure 3.1 shows images of each type.

3.3 Overview of Modeling Approach

To model a 3D paint brush requires developing both a geometric representation and a model for its dynamic behavior. The requirements of an interactive painting system place constraints on the design: the brush dynamics must run at interactive rates and remain stable under all types of user manipulation.

vBRUSH incorporates an optimization-based model for brush dynamics that is capable of simulating the wide variety of brushes that are used in painting with a versatile, multi-spine modeling approach. The geometric modeling approach is a hybrid surface-and-strip representation for the brush head. This approach allows vBRUSH brushes to recreate both the smooth, neat strokes in which individual bristles play a minor role, as well as the more random, scratchy strokes that result from the effects of individual bristles.

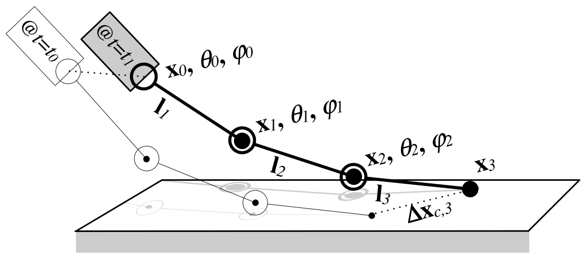


Figure 3.4: *A single brush spine structure shown at two time steps, t_0 and t_1.*

Figure 3.3 shows the geometric structure used to construct each of the brushes described in Section 3.2, as well as their deformation as they make contact with the canvas.

3.4 Brush Dynamics

The transitory behavior of a typical brush subjected to an impulse force is very brief. Stated another way, given an externally applied force system, a brush will reach equilibrium very rapidly. Thus one can approximate the dynamics by solving a static equilibrium problem at each time step.

The basic dynamic model in vBRUSH begins with a spine model composed of several segments as in Fig. 3.4. The bending of each joint i in the kinematic chain is described by two angle parameters, θ_i and ϕ_i. At every step the goal is to minimize the total energy function:

$$E(\mathbf{\Theta}, \mathbf{\Phi}) = E_s + E_f + E_d \tag{3.1}$$

E	Energy of system
E_f	Energy lost to friction
E_s	Spring potential energy
E_d	Energy lost to damping
θ_i, ϕ_i	Joint angles
$\mathbf{\Theta}, \mathbf{\Phi}$	Vectors of all joint angles
μ	Coefficient of friction
$\mathbf{x}_i$	Cartesian coordinates of joint i
$\mathbf{l}_i$	Local coordinates of the end of segment i
$\Delta\mathbf{x}_{c,i}$	Change in contact point i
$F_{n,i}$	Normal force at contact point i
K_i	Stiffness of spring i
D_i	Damping constant of spring i
$\beta(\theta_i, \phi_i)$	Total bending angle of spring i
$\mathbf{x}_p$	A point on planar constraint surface (canvas)
$\hat{\mathbf{n}}_p$	Normal of planar constraint surface (canvas)

Table 3.1: *Summary of mathematical notation for brushes.*

where

$$E_s(\mathbf{\Theta}, \mathbf{\Phi}) = \sum_i K_i \beta(\theta_i, \phi_i)^2/2 \tag{3.2}$$

$$E_f(\mathbf{\Theta}, \mathbf{\Phi}) = \sum_i \mu |F_{n,i}| \|\Delta\mathbf{x}_{c,i}\| \tag{3.3}$$

$$E_d(\mathbf{\Theta}, \mathbf{\Phi}) = \sum_i D_i |\Delta\beta_i| \tag{3.4}$$

subject to

$$(\mathbf{x}_i - \mathbf{x}_p) \cdot \hat{\mathbf{n}}_p \geq 0 \tag{3.5}$$

Please refer to Table 3.1 and Figure 3.4 for a summary of the mathematical notation used in this chapter.

3.4.1 Virtual Work and Optimization

The principle of virtual work can be used to solve for the static equilibrium of a system via minimization. The virtual work done by all external active forces on a mechanical system in equilibrium equals the corresponding change in the total potential energy of the system for any and all virtual displacements consistent with the constraints (Meriam & Kraige, 1992).

$$\delta E_{\text{potential}} = \delta W_{\text{external}} \tag{3.6}$$

or

$$\delta(E_{\text{potential}} - W_{\text{external}}) = 0, \tag{3.7}$$

which is to say the variation of the total energy with respect to an infinitesimal change in configuration is zero. By integrating these equations and analyzing the derivatives in the neighborhood of the critical points, one arrives at the conclusion that stable equilibriums coincide with energy minima.

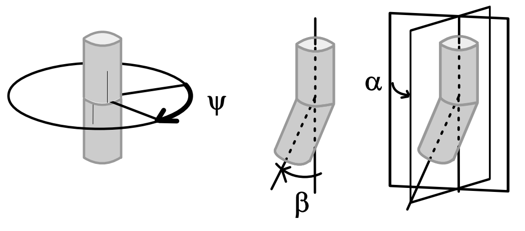
For a constrained system in which one cannot easily express the "consistent displacements" in terms of a minimal number of degrees of freedom, one can express the constraints using Lagrange multipliers and scalar constraint functions of the form $C(\mathbf{q}) = 0$. The augmented objective function to minimize is then given as

$$L(\mathbf{q}) = E(\mathbf{q}) + \sum_{i=1}^{n} \lambda_i C_i(\mathbf{q}) \tag{3.8}$$

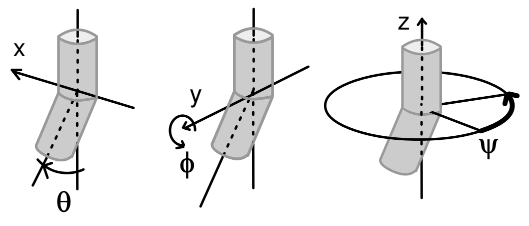
with

$$C_i(\mathbf{q}) = 0, \quad 1 \leq i \leq n. \tag{3.9}$$

Given an optimal solution, $\mathbf{q}^*$, the unknown generalized constraint forces can then be recovered with the expression $\sum_{i=1}^{n} \lambda_i \nabla C_i(\mathbf{q}^*)$. The constraint forces need not be



(a) Euler ZYZ Bristle Angles



(b) Euler XYZ Bristle Angles

Figure 3.5: *Angle parameterizations. I use the θ, ϕ of the XYZ angles because they are singularity-free in the rest configuration ($\theta = \phi = 0$).*

included directly in the energy function because they do no work, and hence do not add or remove energy from the system.

The solution involves the following steps:

1. Move the brush handle from the initial position $\mathbf{b}_0$ to its new position $\mathbf{b}_1$, according to user input from the input device.

2. Move all bristle spines rigidly with handle, not changing any joint angles.

3. Solve optimization problem for each spine, enforcing constraints at this new position.

4. Repeat from Step 1.

Since I first rigidly translate the bristles with the handle, the optimization essentially can be seen as a backward search for the closest configuration consistent with the constraints and in which spring forces are equal and opposite to friction forces.

3.4.2 Spine Kinematics

My kinematic definition of a brush spine differs from that of (Chu & Tai, 2002) in that I use angles from the Euler XYZ angle set (Fig. 3.5(b)) rather than the

ZYZ set (Fig. 3.5(a)). Since the amount of twisting that occurs in real bristles is limited, my parameterization uses only the first two angles, θ and ϕ. All Euler angle parameterizations for rotations have singularities, but where these singularities occur differs. With ZYZ angles a singularity occurs at the point where the angles are all zero, which is the rest configuration of the brush. Singularities are a problem for the optimizer because gradients evaluated numerically at the singularity are essentially noise. With XYZ angles the singularity is on the horizon at 90°, much less likely to interfere.

Given this parameterization, the rotation matrix for a segment $i+1$ with respect to its parent i is given by

$$ {}^{i}\mathbf{R}_{i+1} = \begin{pmatrix} \mathrm{c}\phi & \mathrm{s}\phi\mathrm{s}\theta & \mathrm{s}\phi\mathrm{c}\theta \\ 0 & \mathrm{c}\theta & -\mathrm{s}\theta \\ -\mathrm{s}\phi & \mathrm{c}\phi\mathrm{s}\theta & \mathrm{c}\phi\mathrm{c}\theta \end{pmatrix} \tag{3.10} $$

where c and s are used as abbreviations for sine and cosine of angles. And the full expression for a point $\mathbf{p}$ in terms of its parent frame is

$$ {}^{i}\mathbf{p} = {}^{i+1}\mathbf{p} + {}^{i}\mathbf{R}_{i+1}\mathbf{l}_{i+1}, \tag{3.11} $$

By composing such transforms recursively one can compute the Cartesian positions $\mathbf{x}_i$ of each joint in world space. Derivatives of these expressions are also needed for the optimizer. For more details on the derivations necessary for analytical gradient calculations in the optimizer, please see Appendix A.

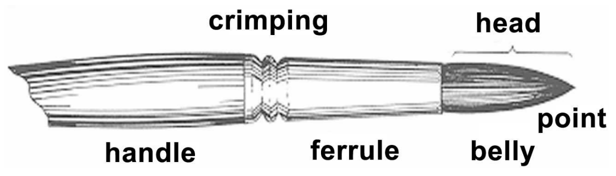


Figure 3.6: *The nomenclature used for parts of a typical brush.*

3.4.3 Spring Energy

The simplest term in the energy function comes from the potential stored in the springs (Equation 3.2). The energy is a function of the total deflection from the vertical. The deflection angle can be computed as

$$\beta(\theta, \phi) = \cos^{-1}(\mathrm{c}\theta\mathrm{c}\phi). \tag{3.12}$$

Several factors make real brushes stiffer near the ferrule than the point (see Figure 3.6). First, the individual hairs in a brush head are thinner at the tip than at the base, making them naturally less stiff near the point. Second, the tight packing of the hairs within the ferrule stiffens the bundle near the base. Finally, in some brushes, not all the hairs extend all the way to the tip of the brush, also leading to less stiffness at the point. To account for this variable stiffness, one can simply set the K_i values higher near the base.

3.4.4 Friction Energy

For the frictional model (Equation 3.3), I use a modified Coulomb law. First, I consider the frictional force to act only on joints that are in contact with the canvas. The force has magnitude $\mu|F_n|$, and this is approximated as a constant over the course of one optimization. I also simplify the problem by assuming that the motion of the joint over the surface can be approximated as a straight line. Given a change in surface contact position $\Delta\mathbf{x}_{c,i}$, the work done by the friction force can be written as shown in Equation 3.3.

In analyzing the work done in a virtual displacement, the virtual work of dissipative forces like friction must be treated as negative. Ignoring E_d for a moment, plugging in $-\delta E_f$ for the external work in Equation 3.7 and E_s for the potential energy one gets $\delta(E_s + E_f) = 0$, as the equation of equilibrium. In order for this to be a *stable* equilibrium it should be a minimum rather than a maximum of the energy function.

Technically, the frictional function does not meet the requirements for most numerical minimization techniques, since E_f is not differentiable at the point where $\Delta\mathbf{x}_{c,i} = 0$, and optimization routines rely on the differentiability of the objective function. This characteristic can cause difficulty for some optimization techniques; however, an SQP solver that approximates the second derivative of E with finite differences (a BFGS Hessian approximation) works well enough in practice. When evaluating gradients, vBRUSH reports 0 as the friction gradient at the apex of the cone. The BFGS finite difference approximation of the Hessian tends to smooth out the second derivative locally, which at worst allows the contact point to slip slightly when friction would otherwise cause it to stick. The fix is to use set-valued derivatives (*sub-differentials*) instead of ordinary differentials, but this introduces high overhead.

Stiction

The friction model above does not account for the frequently observed stiction effect in which the coefficient of static friction, μ_s is greater than kinetic or sliding friction μ_k. In order to incorporate this effect, I propose a simple solution. When in the sticking state, first solve the optimization problem using μ_s. If the solution indicates $\Delta\mathbf{x}_{c,i} > 0$, then switch to the sliding state and rerun the minimizer using μ_k. When in the sliding state, if $\Delta\mathbf{x}_{c,i} < \epsilon$ then switch to the static state for the start of the next optimization. Hysteresis can also be used in determining the transition thresholds.

Anisotropy

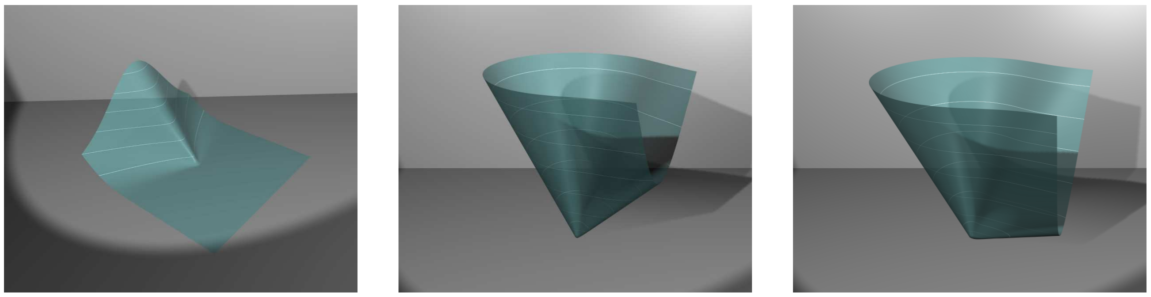
The tips of bristle hairs are more likely to get caught in the tooth of the painting surface when pushed as opposed pulled over the paper. vBRUSH incorporates this effect simply and efficiently by multiplying an anisotropy term in the calculation of the friction. I have devised a simple anisotropy term that can be combined with E_f. The advantages of this function are that it has C^1 continuity and its analytical derivatives can be efficiently computed. Let $\hat{\mathbf{x}}$ represent the unit vector $\mathbf{x}/\|\mathbf{x}\|$ in what follows. The modified anisotropic energy function is given by:

$$E_f = (1 - \eta)\mu|F_n|\|\Delta\mathbf{x}_{c,i}\| \tag{3.13}$$

where

$$\eta = C_\eta \max\left(0, \mathbf{d}_p \cdot \frac{\mathbf{\Delta x_{c,i}}}{\|\mathbf{\Delta x_{c,i}}\|}\right)^k, \; C_\eta \in [0, 1]. \tag{3.14}$$

$\mathbf{d}_p$ is the preferred direction, i.e. the direction of minimal resistance (the "pull" direction) for the bristle. This expression resembles that of the intensity of a Blinn-Phong of a specular highlight, and seems to work quite well as a model for anisotropic friction, as well. Examples of the overall Coulomb friction energy for different values of C_η are shown in Fig. 3.7. The C_η and k constants give one an intuitive way to control the anisotropy. $C_\eta = 0$ recovers the isotropic case, and $C_\eta = 1$ removes all friction in the preferred direction. The k determines how sharply focused anisotropy will be, just as it does in computing specular highlights.

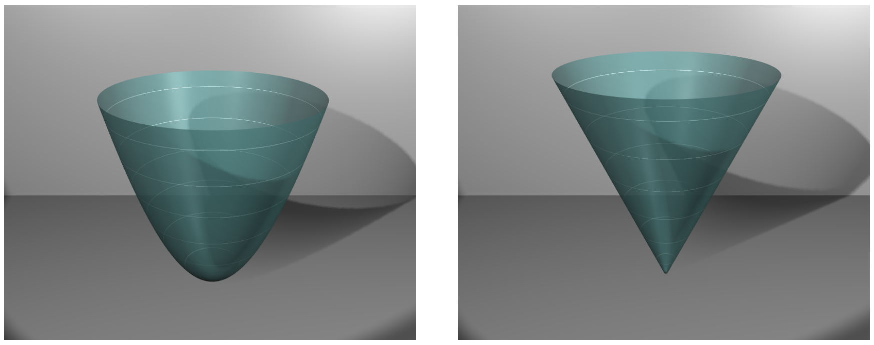


(a) The anisotropic term by itself.

(b) $C_\eta = 0.5$

(c) $C_\eta = 0.8$

Figure 3.7: *Anisotropic Coulomb frictional energy function. The function (a) is subtracted from the function in Fig. 3.8(b) to get (b). (Note, orientation of the the preferred direction is reversed in (a) to better display the geometry of the anisotropy).*



(a) Spring potential, E_s

(b) Friction work, E_f

Figure 3.8: *Some energy terms that make up the objective function. Note that the horizontal axes of (a) are joint angles, while in (b) they are X and Y components of $\Delta\mathbf{x}_{c,i}$ for one joint.*

3.4.5 Damping and Plasticity

Another type of friction that affects some brushes significantly is internal friction and drag between wet bristles. vBRUSH models this as a constant resistance in joint space:

$$\mathbf{F}_{d,i} = -D_i \mathrm{sgn}(\Delta\beta_i) \tag{3.15}$$

and incorporates it as part of the overall energy as shown in Equation 3.4.

The effect of adding this internal friction term is that a deformed brush does not return all the way to its starting point, since the damping forces are greater than the spring bending forces for small angles. In effect, this works as a simple model for brush plasticity. Plasticity is achieved by other means in (Chu & Tai, 2002), but using joint friction is both simple to implement and perhaps more closely related to the actual physical mechanism behind brush plasticity.

3.4.6 Derivatives

Most optimization methods, including the SQP method I use for brush simulation, work best when at least first derivatives of the objective function and constraints can be computed analytically. If they are very expensive to compute, it is possible to estimate derivatives with finite differences, but this is not as reliable, and it requires multiple evaluations of the objective function to approximate one derivative. For brush energy optimization, the derivatives of Equation 3.1 with respect to all the joint parameters are needed, and these can all be expressed in closed-form. Computationally, the analytical derivative expressions are similar in cost to the objective function itself, and contain many common subexpressions, which can be reused for greater efficiency. The full list of derivatives needed for implementing the optimization routines is given in Appendix A.

3.4.7 Constraints

To prevent the joints of the brush spine from penetrating the canvas, vBRUSH subjects each to an inequality constraint that expresses that the Cartesian joint position must be outside the plane of the canvas, as given in Equation 3.5. Note that $\mathbf{x}_i$ is a nonlinear function of θ_j and ϕ_j for $1 \leq j \leq i$, given by the forward kinematic Equations 3.10–3.11.

The SQP minimizer I use relies on an active set approach, in which inequality constraints are treated as equality constraints while active, and simply ignored when not active (Agrawal & Fabien, 1999). This approach is efficient in that the dimensionality of the augmented objective function (Equation 3.8) is reduced when few constraints are active, leading to smaller matrix equations to solve.

3.5 Geometric Modeling of Brushes

The vBRUSH geometric model for brushes can take advantage of both an explicit surface representation as in (Baxter et al., 2001; Chu & Tai, 2002; Saito & Nakajima, 1999) but also a strip based method for representing the brush as individual hairs. A brush can use either one of these or both. The surface-based representation has the advantage of giving a solid, consistent footprint; however, it is difficult to capture bristle spreading effects and the stippling effects created by individual bristles with just a surface-based model.

3.5.1 Subdivision Surface

For the surface-based representation, I use a subdivision surface, the location of whose control vertices are tied to the motion of the underlying dynamic brush skeleton. The brush skeleton consists of a set of the optimization-based structures described in the previous section. In my initial work on brush modeling (Baxter et al., 2001), I used the interpolating Butterfly subdivision scheme since this made it easier to manually place control vertices relative to the skeletal structure in order to achieve the general deformation desired. However, interpolating subdivision surfaces can easily develop unnatural high-curvature kinks. The degree of continuity achieved by approximating schemes is generally superior to that of interpolating schemes.

In this work, I propose to use an approximating Catmull-Clark surface instead of an interpolating Butterfly surface. The challenge of placing appropriate control vertices to achieve a desired shape has been dealt with by creating an export tool for a popular, free 3D modeling package called Blender(Blender, 2004). This package enables users to visually and interactively create brush surfaces of any shape, and my exporting routines allow users to specify the desired physical properties (stiffness, damping, and friction).

The control vertices that determine the shape of the subdivision surface are placed using standard matrix skinning techniques. The skinning weights and brush spines can be set up from within Blender using its built-in skeletal animation tools. My exporter converts Blender's data structures into those used by vBRUSH.

3.5.2 Bristle Strips

A typical real brush head can be composed of a few thousand hairs. The most accurate model of a brush would involve simulating each one of these individually, including full hair-to-hair collision and response, but this is not currently feasible in real time. Nevertheless, the effects of individual bristles are important to capture. Fortunately, the motions of individual bristles show a large amount of coherence—neighboring bristles tend to move more or less in the same direction—so vBRUSH takes advantage of this by simulating only a few brush spines (generally less than ten) that form a brush skeleton, and interpolating as many as hundreds of other bristles from the motion of those spines.

Geometrically, each hair is most like a thin, tapered tube; however, tubes require many vertices to specify and it is not clear that the added complexity would add any character to the strokes produced. Instead I use strips of quadrilaterals as the primitive, which require only two vertices per joint and are quick to render in modern hardware. An alternative would be to use line strip primitives, which would only require one vertex per joint, but line widths on graphics hardware can only be specified in image space as an integral number of pixels. Thus the density of a set of line strips changes depending on the resolution of the canvas being painted upon. If hardware allowed for true lines with geometric widths, this would be the most efficient representation.

Paint Transfer:

Paint transfer will be described more fully in the next chapter, but in order to achieve smooth paint transfer with bristle strips, there are a few special considerations that affect how the geometry is managed.

First, when rendering the brush footprint, the flat side of the strips should always face the canvas. In this respect the strips are like viewer-oriented impostors or billboards, with the “viewer” being the canvas. vBRUSH performs this step on the CPU during every paint transfer step. A small vertex program on the GPU could also be used to perform this computation.

The second issue is rendering order. In a real brush, hair-to-hair collision interactions prevent hairs on one side from collapsing through to the other side of the brush. In order to avoid the cost of explicitly calculating this $O(n^2)$ interaction, vBRUSH instead determines an approximate back-to-front rendering order with respect to the canvas using a simple heuristic. It first calculates a principal bend direction for the brush head as the average of all the brush spine tip deflection vectors: $\sum_{i=1}^{N} \Delta\mathbf{x}_{i,\text{tip}}/N$ (see Fig. 3.9). It then sorts the strips by their root positions along the principle bend direction projected onto the brush cross-sectional plane. In the difficult case when

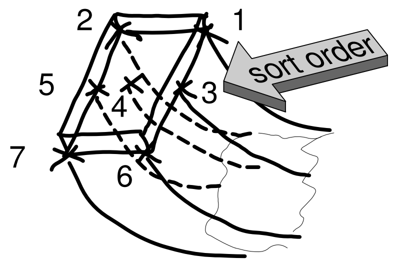


Figure 3.9: *Determination of rendering order for paint transfer. The average bend direction of the spines determines the order in which bristles are sorted and rendered. In the example above the seven bristles shown would be rendered according to the numbering shown. This ordering ensures the brush will maintain consistent separation between front and back sides of the brush during paint transfers.*

all the hairs are bent in the same direction, and many are overlapping, this approach performs quite well. When the spines are splayed out in different directions the ordering may not be consistent with actual depth, but in that case there is little hair-to-hair collision to begin with, so rendering order is not important.

Bristle Dynamics Interpolation:

In creating a new brush model, positioning hundreds or thousands of bristle strips would be tedious, so I have devised an algorithm for automatic random placement. First, I compute an oriented bounding box (OBB) around the root points of the brush spines. From the length of the edges of the OBB, I determine whether sampling should be performed over a 1D or 2D space. If 1D (i.e. the spine roots are all contained in a very narrow bounding box), then I parameterize the best fit line and place strip roots randomly along that line. If 2D, I compute a Delaunay triangulation of the brush spine roots and then place strip roots randomly within those triangles.

At run-time, the geometry of an interpolated strip is computed by a convex weighted sum of spine positions. Each spine is parameterized as $\mathbf{p}_i(s), 0 \leq s \leq 1$ such that $\mathbf{p}_i(0)$ is the root of the *ith* spine and $\mathbf{p}_i(1)$ is the tip. Then an interpolated bristle is given by:

$$\mathbf{p}_{\text{interp}}(s) = \sum_i w_i \mathbf{p}_i(s) \tag{3.16}$$

where $\sum_i w_i = 1$ and $w_i \geq 0$.

Given a randomly chosen root location $\mathbf{p}_{\text{interp}}(0)$, one needs to determine an appropriate set of non-negative weights w_i, which sum to unity and result in the desired position. Since vBRUSH already uses a constrained SQP solver, it is convenient to use minimization to compute this set of weights as well. One can encode the preference that nearby spines should have higher weights than distant spines in the objective function.

I tried several objective functions and observed that minimizing the following weighted least-squares function for w_i yields good results:

$$\sum_i \|\mathbf{p}_{\text{interp}}(0) - \mathbf{p}_i(0)\|^2 w_i^2 \tag{3.17}$$

subject to

$$\begin{aligned} \sum_i w_i &= 1 && (3.18) \\ w_i &\geq 0 && (3.19) \\ \mathbf{p}_{\text{interp}}(s) &= \sum_i w_i \mathbf{p}_i(s) && (3.20) \end{aligned}$$

The coefficients on the w_i are small for spines *near* the interpolated bristle and large for spines *far away*, thereby leading to the opposite trend in the w_i values themselves. That is, an interpolated strip is influenced most by the spine closest to it.

3.6 Implementation and Results

I have tested the implementation of the vBRUSH brush model on a 1GHz Pentium IV™ desktop, and incorporated it with the paint models to be presented in the next three chapters. I implemented an object-oriented C++ SQP optimizer based on the C code accompanying (Agrawal & Fabien, 1999). I have found 5 iterations of the optimizer to be sufficient for a 3-joint brush spine. This takes about 100 microseconds, allowing the simulation of approximately ten spines interactively. The cost of the interpolation bristles is very small, allowing the use of up to 256 bristle strips interactively without difficulty. The main limiting factor is the amount of texture memory used for mapping paint attributes onto the bristles. The memory and system bandwidth consumed by large brush textures can lead to a significant bottleneck; however, with the rapid growth of GPU performance, I conjecture that this bottleneck will be alleviated in the near future.

As a test of the effectiveness of the brush model I have attempted to recreate stroke samples included as training materials for a popular decorative painting technique called One-Stroke™ (http://www.unctv.org/onestrokepainting/). The One-Stroke techniques emphasize using the flexibility of the brush and its complex loading to create organic shapes such as flowers and birds quickly and with a high degree of realism using very few strokes. This type of painting was very difficult with previous painting programs that lacked complex brush loading and highly flexible deformable 3D brushes.

In Figure 3.10, I compare the results of recreating the strokes with the vBRUSH brush model versus results with the model presented in (Baxter et al., 2001). As can be seen in Figures 3.11–3.13 the new brush model is capable of creating detailed bristle marks that cannot be captured by a surface representation alone.

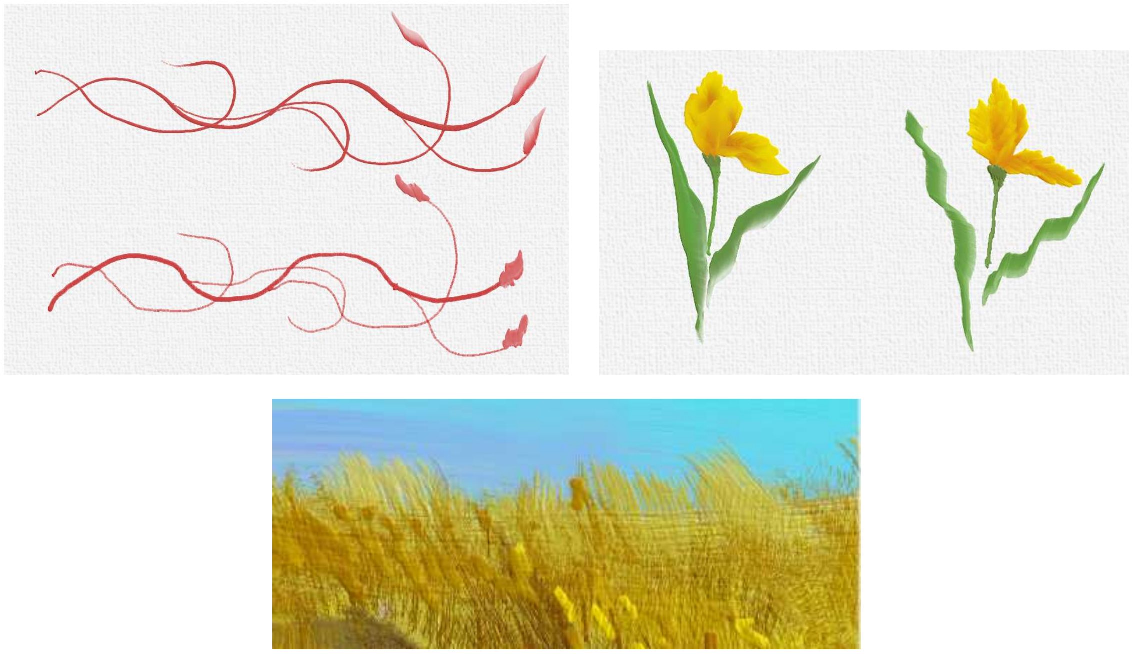


Figure 3.10: (Top row) *A comparison of strokes made with the optimization-based brush model versus that of (Baxter et al., 2001). Note the smoother path and width variations in the new strokes (top row and left side, respectively).* (Bottom) *These strokes are impossible to achieve with the prior brush model.*

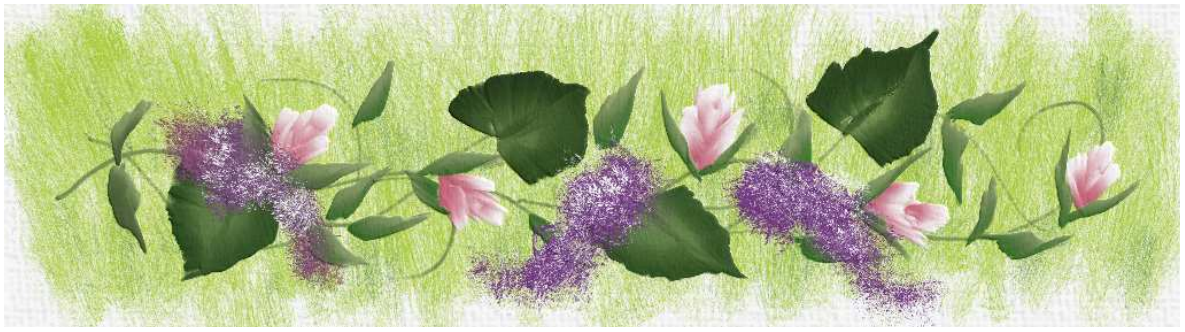


Figure 3.11: *A painting created using both surface and strip based brushes.*

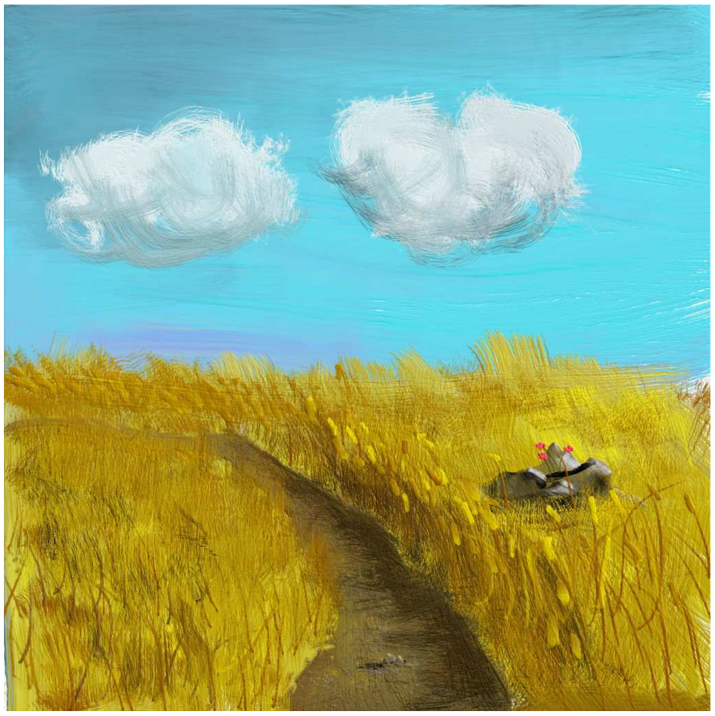


Figure 3.12: *A painting created with my new strip brush models. Notice the detailed, scratchy bristle marks in the clouds and grass.*

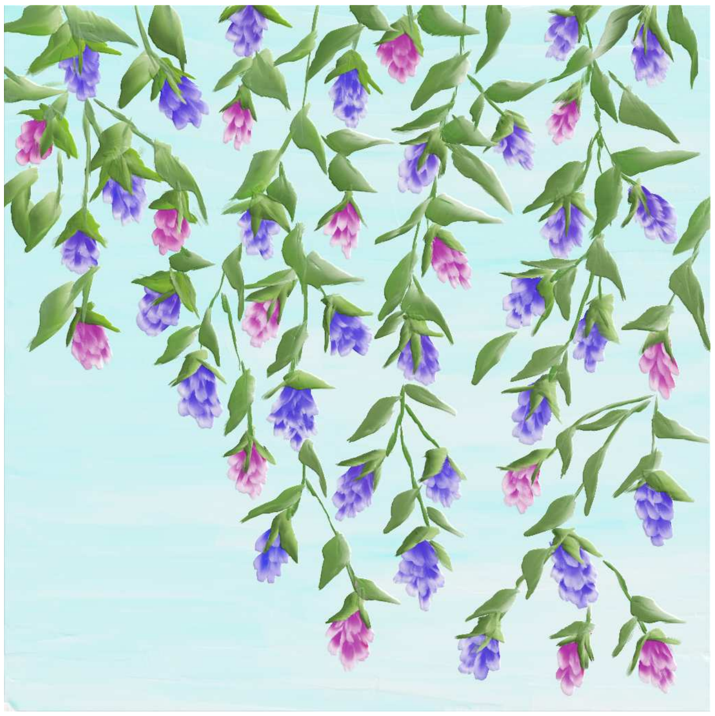


Figure 3.13: *A painting created using my new brush models.*

3.7 Limitations

The vBRUSH brush modeling and simulation technique outlined in this chapter is very flexible and has been successfully used to model a wide variety of brushes. However there are some limitations to the current implementation and some important areas for possible future improvements. First, a few fairly straightforward extensions have not yet been implemented. Springs with a bent resting configuration are not currently possible, nor are collisions with more than a single plane. Adding these would extend the generality of the current implementation, and should require little extra numerical machinery.

An interesting area for future work is to implement automatic on-the-fly generation of bristle strips based on the deformation of the brush. This would allow for better smooth to scratchy transitions. The challenge is to create and destroy the bristles strips in such a way that they do not create popping artifacts, perhaps by alpha blending them in and out. Another area for future work is to somehow capture the effects of bristle-to-bristle collisions, preferably without resorting to an $O(n^2)$ all pairs collision response algorithm. This could be achieved by adding a brush volume preservation term to the optimization that would push bristles outward when they start to collapse.

3.8 Summary

In this chapter I have presented a versatile model for virtual brushes that can reproduce the wide range of effects and styles needed by the digital painter. The key features of this model are:

- Dynamic brush deformation using quasi-static optimization based on the principle of virtual work.

- The combination of subdivision surfaces and strips, which allows for both smooth, detailed strokes, as well as rough scratchy brush work in which individual bristle details can be seen.
- Weighted interpolation of strip bristles based on a small number of physically-simulated spines, which has been shown sufficient to yield a realistic overall deformation of the brush head.
- Brushes with hundreds of hairs that can easily be simulated interactively.
- Brush design accessible to non-programmers using a custom export script for an off-the-shelf 3D modeling package.

Chapter 4

dAb Paint: A Simple Two-layer Paint Model

If I could say it in words there would be no reason to paint.
— Edward Hopper

Having presented a method for modeling the paint brushes used in traditional painting, I now proceed to the second major technical component of this dissertation, techniques for modeling paint. I present three different paint models in this and the following two chapters. As I describe the first paint model, *dAb*, in this chapter, I will also be introducing a few fundamental concepts, like “bi-directional transfer” and “footprint generation”, that will be referred to again in the later chapters without further explanation.

As discussed in the overview in Chapter 2, all three paint models share the goal of serving as a plausible digital stand-in for true physical paint. The three paint models differ, however, in the extent to which they trade off physical principles for speed of execution. As a result, they differ greatly in specifics of design and implementation.

There are two main categories of information presented in this chapter, both of which were first presented in (Baxter et al., 2001), though the coverage of the topics here has been expanded significantly. On the first subject, footprint generation, I present a general treatment of techniques relevant to systems in which texture-mapped virtual 3D objects are to be used as mark-making tools. In this dissertation that means virtual paint brushes, but it could just as easily apply to pencils, or crayons, or chalk, etc. In all of these cases there are similar operations necessary to determine the extents and attributes (color, density, etc.) of the marks made on the paper or canvas, as well as the extent and nature of the effect on the mark-making implement itself. For instance, soft implements like pencils, charcoal, or crayons have a tendency to erode away in the vicinity of contact, and it may be desirable to model this change in the drawing implement itself. A paint brush can pick up paint from the canvas in areas of contact. So a fundamental task in any of these systems is to compute these contact areas, or footprints, not only on the canvas but also on the mark-making implement itself.

The second subject of this chapter is the specific nature of the dAb paint model itself. The dAb model is a simple, efficient two-layer model, suitable for interactive use on most computer systems, but which is still capable of capturing a variety of paint-like effects.

4.1 dAb Features

The dAb paint model incorporates partial drying and translucency, conservation of volume, and a bi-directional paint transfer algorithm. It supports the following operations and techniques expected from acrylic or oil painting, while maintaining interactivity on a wide range of hardware:

- **Blending** – Mixing of multiple pigments to obtain the desired color.
- **Bi-directional transfer** – Transferring paint not only from the brush to canvas, but also back from the canvas to the brush.

- **Complex brush loading** – Filling different portions of the brush head with varying amounts of different colors.
- **Controlled drying** – Controlling the blending of new paint onto previous layers by allowing paint to partially dry.
- **Glazing** – Painting with translucent layers of colors over previous layers.
- **Impasto** – Painting with an appearance of thickness.

The remainder of the chapter will describe in detail how these effects are achieved.

In terms of implementation, the dAb model uses the main system CPU to perform all the model-specific paint blending, but uses graphics hardware (the GPU) to compute footprints. In designing the dAb paint model, care has been taken to reduce memory consumption, and to avoid reliance on advanced hardware features like GPU fragment programs. This was partly out of a desire for portability and partly just the simple fact that at the time of dAb's initial development (2000), no commodity graphics card was capable of programmable shading or 32-bit color channels. Many GPUs sold today still lack those features, or run them too slowly to be useful, especially those used in mobile computing devices. Of the three paint models in this dissertation, dAb is the only one that runs interactively on most current laptops.

4.2 Mathematical Description of Contact Footprints

The first major step in painting a stroke is to determine the contact area between the brush surface and the canvas, and to establish a mapping between the texture space of the brush and the texture space of the canvas. In this section I describe mathematically what is meant by a footprint and develop notation that will allow description of the necessary operations concisely.

Let $\Omega \subset \mathbb{R}^3$ be the surface of an object in 3-space for which there exists some two-degree of freedom parameterization, $\Omega(\alpha, \beta)$ (i.e. Ω is a 2-D manifold embedded in 3-space rather than a volume). Then a texture parameterization can be thought of as establishing a mapping $\mathcal{T} : \Omega \rightarrow \mathbb{R}^2$, from points on the object to 2D texture coordinates[1]. These texture coordinates can then be used to index one or more attribute maps[2] $\mathcal{A} : \mathbb{R}^2 \rightarrow \mathbb{R}^N$ to determine values associated with that point on the surface. Given texture mappings $\mathcal{T}_B$ and $\mathcal{T}_C$ and attribute mappings $\mathcal{A}_B$ and $\mathcal{A}_C$ for the brush surface and canvas surface, respectively, the ultimate goal is to update both the brush's and the canvas' attribute maps in such a way that the final result mimics the physical real-world transfer of paint between brush and canvas.

First, one must determine which portions of the brush and canvas are in contact. Let the brush surface be given by $\Omega_B \subset \mathbb{R}^3$ and canvas surface by $\Omega_C \subset \mathbb{R}^3$. Then one might attempt to define the contact surface as simply $\Omega_B \cap \Omega_C$. The brush simulation is designed to deform the brush surface when it comes in contact with the canvas surface so they do not interpenetrate; however, due to various numerical issues it is best to allow some tolerance in determining contact. In practice I define a non-commutative approximate intersection operator such as, $\cap_\epsilon$, defined as $A \cap_\epsilon B = \{x : x \in A, \exists y \in B \text{ s.t. } \|x - y\| < \epsilon\}$, i.e. all points of A that are less than ϵ distance from B. One can then define the portion of the brush in contact as $\Omega_{BC} = \Omega_B \cap_\epsilon \Omega_C$ and similarly the portion of the canvas in contact as $\Omega_{CB} = \Omega_C \cap_\epsilon \Omega_B$.

It is now possible to precisely define exactly what is meant by the contact footprint. In fact there are two contact footprints of interest, the footprint in canvas texture

[1]In computer graphics, the parameterization is usually defined by associating explicit texture coordinates with vertices of triangles and determining the texture coordinates over the remainder of the 3D surface by barycentric interpolation. The end result is the same, however: every point on the model surface is assigned a 2D texture coordinate.

[2]Or texture maps, in common parlance, though the map may not necessarily encode a surface's texture or color. It could encode translucency or viscosity or any other set of attributes.

space, defined by $\mathcal{F}_C = \mathcal{T}_C(\Omega_{CB})$, and the footprint in brush texture space, defined by $\mathcal{F}_B = \mathcal{T}_B(\Omega_{BC})$. The canvas, Ω_C, in my implementation of the dAb model, is a simple rectangular subset of $\mathbb{R}^2$, like $[0, X] \times [0, Y]$. In this case, texture coordinates are related to geometric coordinates by a simple translation, scale and axis-aligned orthographic projection, $\mathcal{T}_C(x, y, z) = (k_1(x - x_0), k_2(y - y_0))$. Furthermore, the appearance of the brush's footprint in canvas texture space will coincide roughly with the the intuitive notion of a "footprint," that is, the 2D shape of a 3D object impressed on a surface. Of course, it is not technically necessary for the "canvas" to be a flat surface; however, this restriction makes implementation much simpler. Also note that for more complex mapping functions, $\mathcal{T}$, it is not necessary that footprints be connected (in the set theory sense) even if the contact geometry Ω that generates them is a connected set. This is because the texture mapping itself may be discontinuous. That is, it is possible that $\exists p_1, p_2 \in \Omega$ such that $\|p_1 - p_2\| < \epsilon$ but $\|\mathcal{T}(p_1) - \mathcal{T}(p_2)\| > k\epsilon$ for all finite $k \in \mathbb{R}$. (In this case p_1 may be located along a texture seam that intersects the segment connecting p_2 to p_1.) Topologically, such situations can be unavoidable. For instance if the brush is a sphere it is well known that no singularity-free parameterization of the surface can exist.

Now, in order to determine what sort of paint transfer will take place between the brush and canvas, it is necessary to compute a mapping that gives the brush texture coordinate that is located at the point in Ω_B closest to each point in Ω_{CB}, the contact surface on the canvas. To this end, let us define a function that takes canvas texture coordinates and maps them to the closest brush texture coordinates: $\mathcal{T}_{CB} : \mathbb{R}^2 \to \mathbb{R}^2$. Given this mapping function, one can take a point from the canvas footprint, $p \in \mathcal{F}_C$, and determine the brush attributes in contact at that point, by using $\mathcal{A}_B \circ \mathcal{T}_{CB}(p)$, where '$\circ$' denotes function composition.

Likewise, to determine how paint will be transferred from canvas to brush, one needs $\mathcal{T}_{BC} : \mathbb{R}^2 \rightarrow \mathbb{R}^2$ that maps from brush texture coordinates to the corresponding canvas texture coordinates, allowing one to compute the canvas attributes at a point $p \in \mathcal{F}_B$ with $\mathcal{A}_C \circ \mathcal{T}_{BC}(p)$. The problem of determining brush and canvas footprints thus reduces to computing $\mathcal{T}_{CB}$ and $\mathcal{T}_{BC}$, or acceptable approximations thereof.

In practice, all of dAb's attribute maps are discretely sampled texture maps rather than continuous functions, but the problem of determining mappings remains similar, except that values such as $\mathcal{A}_C \circ \mathcal{T}_{BC}(p)$ need to be determined by interpolation using the nearest available samples in $\mathcal{A}_C$.

The main benefit of using a polygonal brush representation and texture mapping to assign paint information is that it enables one to generate the contact footprint simply and quickly using rasterization hardware. The GPU graphics pipeline is designed specifically to generate rasterized 2D projections of texture-mapped 3D polygons as rapidly as possible. The problem of footprint generation at a given instant can be viewed simply as such a projection problem. This process will be described in Section 4.5.1. The reverse problem of mapping canvas attributes to brush texture space is a bit more complex, though still fairly straightforward. The reverse mapping is explained in Section 4.5.3.

4.3 Paint Stroking Algorithms

The brush is simulated as described in Chapter 3. The result of the simulation is a representation of the brush surface at each instant in the form of either a connected polygonal mesh or a set of individual polygonal strips. In either case, the paint loaded on the brush is mapped directly onto the polygons via standard texture mapping (Catmull, 1974). The paint on the canvas is also stored as texture maps. The actual representation of the paint and the detailed contents of the texture maps will be described in the next section.

Although the brush geometry is only available at discrete time intervals, a painted stroke should be continuous with no gaps. An obvious solution to prevent gaps is to perform some kind of interpolation between successive brush geometries to connect the footprints they generate. There are several ways to perform this interpolation. I have developed a faster method (Algorithm 1), and a more accurate method (Algorithm 2). Note that this interpolation problem is more complex than that handled by most existing painting programs for two reasons: 1) The footprint shape is not constant, nor is it even taken from a set of predetermined 2D shapes. Each footprint is generated uniquely from intersection with a dynamically deforming 3D brush. 2) The brush color is not a single value—the brush paint is represented as a distribution of color and attributes over the surface of the brush, so the color and attributes must also be interpolated. The faster method (Algorithm 1) provides strokes that appear to be connected as long as the footprint geometry does not change dramatically between two time steps. Algorithm 2 is more computationally intensive, but generates connected strokes even for the difficult cases of a brush with large rotational motion or large change in deformation between time steps. Examples are shown in Figure 4.1.

Pseudocode of the high-level algorithms for the two methods is given below. In the faster method, a fixed 2D footprint is first generated, and then that fixed footprint is blended along the stroke trajectory. In contrast, in the higher quality method, the actual brush geometry is linearly interpolated between two sampled positions, and the contact footprint is recomputed every step along the stroke path. A natural extension, though one not currently implemented, would be to use a higher order interpolation, e.g. cubic splines, to connect successive brush positions.

Note that if computational cost were not a factor, the ideal interpolation scheme

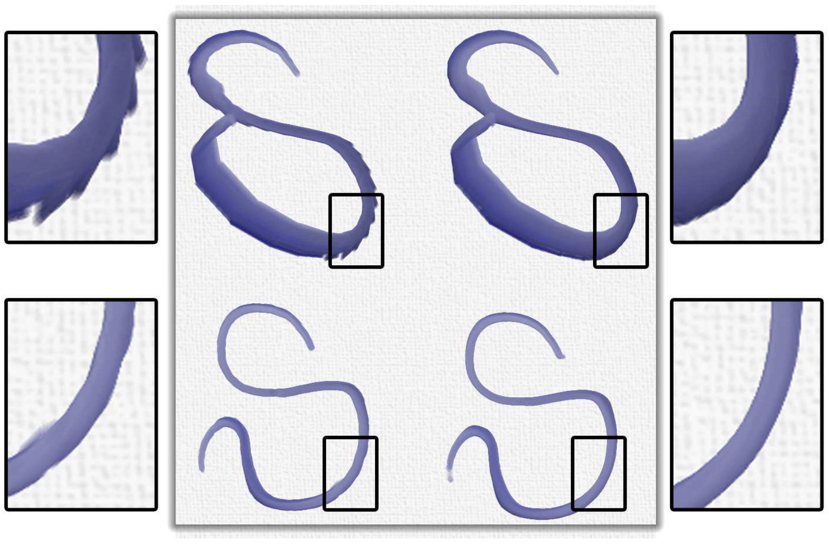


Figure 4.1: *A comparison of the same stroke paths as drawn by the two stroking algorithms. The strokes on the left were painted with Algorithm 1, those on the right with Algorithm 2. The top two strokes were painted with a fast motion, while the bottom two were painted more slowly. The insets show a 3× zoom of the indicated portions of the strokes.*

would be to execute the full physically-based brush deformation simulation at successively finer time intervals until the motion of all points on the brush was less than one pixel in every time interval. This, however, is not computationally feasible with the current brush dynamics model.

The following sections will describe the steps of these algorithms and the implementation of them in more detail.

4.4 Paint Representation

The two-layer dAb model separates each paint surface into a thinner outer layer and a potentially much thicker under layer, referred to as the 'surface' and 'deep' layers, respectively (See Figure 4.2). Both the canvas and brush surface layers are considered to be completely *wet*. For the canvas, the deep layer represents the paint that is completely *dry*, while for the brush, the deep layer represents the reservoir of wet paint contained

```
STROKE-FASTER(brush, t1, t2)
1  Determine mp1, the anchor point of brush[t1]
2  Determine mp2, the anchor point of brush[t2]
3  Generate fp, the footprint of brush[t2]
4  for pos ← mp1 to mp2      ▷ Step so pos moves 1 canvas texel at a time
5      do Blend fp with canvas at pos
   end
7  Update brush textures with blended fp
```

ALGORITHM 1: *The faster stroking algorithm. Given a brush time sequence and two times, this function computes one fixed footprint and blends that shape along the stroke path.*

```
STROKE-BETTER(brush, t1, t2)
1  d ← max distance between corresponding points in brush[t1], brush[t2]
2  for $\alpha$ ← 0 to 1      ▷ Step so $brush_\alpha$ moves 1 canvas texel at a time
3      do
4          $brush_\alpha$ ← LINEAR-INTERP($\alpha$, brush[t1], brush[t2])
5          Generate $fp_\alpha$, the footprint of $brush_\alpha$
6          Blend $fp_\alpha$ with canvas
7          Update brush textures with blended $fp_\alpha$
   end
```

ALGORITHM 2: *The higher-quality stroking algorithm. Given a brush time sequence and two times, this function interpolates the 3D geometry of the brush between the two times, regenerating the transfer footprint repeatedly each step of the way.*

within the bristles. The wet surface layers are where all paint transfer between brush and canvas occurs. Each paint surface (either canvas or brush) is represented as two color textures plus an additional attribute texture. Each of the textures is stored as 24-bit RGB or 32-bit RGBA (See Table 4.1).

The dAb model moves paint between layers in three different cases, as illustrated in Figure 4.2. First, paint transfer between the canvas surface and brush surface layers occurs upon contact between the two. This is the main bi-directional transfer operation that enables painting. Next, as the surface layer of the brush becomes depleted during

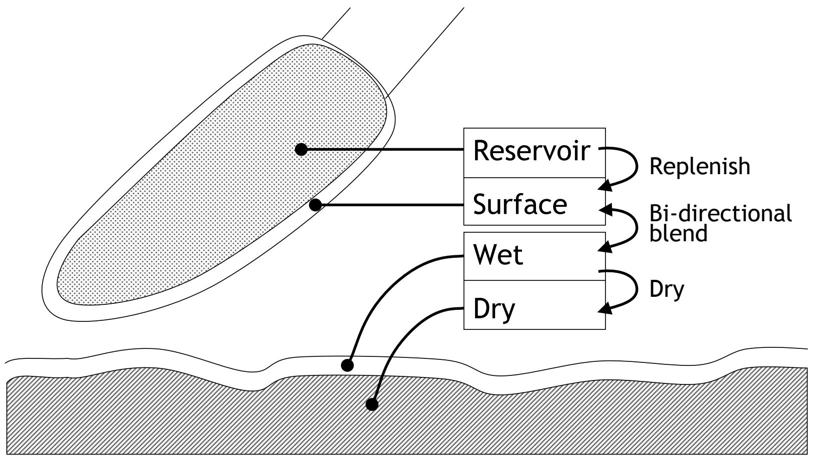


Figure 4.2: *The layers used to represent brush and canvas in the dAb paint model, and the operations that move paint between layers. Drying moves wet paint from the Wet layer to the Dry layer of the canvas. Bi-directional blending moves paint back and forth between the outer layers of the brush and canvas. When the brush Surface layer is low on paint, it is replenished from the Reservoir layer, unless the Reservoir layer is empty.*

painting, it is refilled from brush's deep reservoir layer. Finally, drying the canvas (Section 4.7) causes paint to move from the canvas's surface layer to its dry layer, and this occurs either on a timed interval or as requested by the user.

One additional RGB texture is used to cache the final rendered canvas, so that it can be redisplayed quickly. For efficiency, the entire canvas is divided into tiles of size 32×32, and only tiles that have been modified by a painting stroke or by drying need to be recomputed and uploaded to the graphics card.

Using 8-bit-per-channel textures to store all the paint information allows the dAb paint model to work with the widest range of graphics hardware possible. The volume of paint in each layer is stored in one channel of a 24-bit RGB attribute texture (see Table 4.1). Since the paint volume of the brush surface is stored in just 8-bits, it can only hold 255 "units" of paint. To allow the brush to hold more paint, each unit of volume in the deep layer is treated as 32 units of surface paint. In order to allow values

Channel	**Canvas**	**Palette**	**Brush**
Red	Wet layer volume	Wet layer volume	*unused*
Green	Dry layer embossing	Shape mask	Reservoir volume
Blue	Dry layer relative heightfield	Region code	Surface volume
Alpha	N/A	N/A	Footprint stencil

Table 4.1: *Attribute textures used in the dAb model and the contents of each channel.*

greater than 255 in the deep layer of the canvas, its volume (or height) is encoded using a special relative height field scheme, which is described in Section 4.8.

4.5 Details of the Paint Stroking Algorithms

This section steps through the details of the core operations performed in Algorithms 1 and 2 in the order in which they are performed. These main steps are:

1. Generating the footprints on the canvas;
2. Bi-directional paint transfer and blending;
3. Updating the brush textures.

.

4.5.1 Generating Footprints on the Canvas

The footprint generation step (Algorithm 1, line 3; Algorithm 2, line 5) takes a polygonal, texture-mapped 3D brush surface and determines the image-space 2D contact region for paint blending. In other words, this step computes $\mathcal{A}_B \circ \mathcal{T}_{CB}$ described in Section 4.2. Assuming a planar canvas, footprint generation can be accomplished efficiently by using graphics hardware to rasterize the brush using its three different texture maps: surface, deep, and attribute. Specifically, the brush is rasterized using an orthographic projection, with the viewpoint located above the canvas, and the view

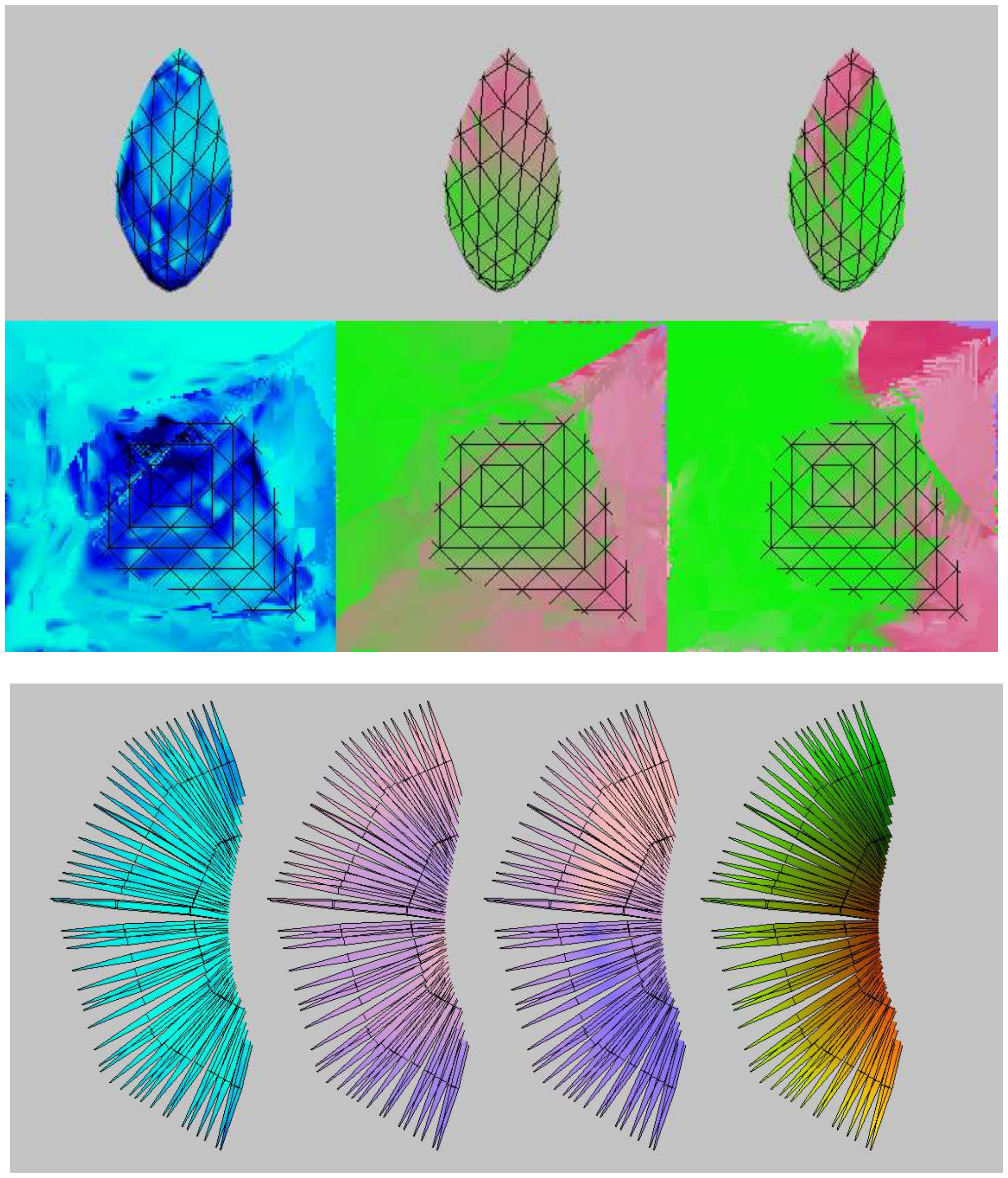


Figure 4.3: *The footprint attribute textures as rendered to the canvas texture space. The projected polygons of the brush mesh are shown in black, though these are not rendered normally in the process of the algorithm. The top row shows a mesh brush's footprints for a brush loaded with a mixture of pink and green paint. From left to right: brush attributes, surface color, and reservoir color. For brush attributes, green indicates reservoir volume and blue indicates surface volume (Table 4.1). The second row shows the same brush's full textures. The third row shows the footprint textures of a non-mesh brush, including a fourth footprint containing brush texture coordinates.*

direction parallel to the canvas' outward normal, $\mathbf{n}$. Any portion of the brush within ϵ of the canvas plane is considered to be in contact. To restrict the rasterization to that portion of the brush geometry, a near clipping plane is set parallel to the canvas, $\epsilon\mathbf{n}$ units in front of the canvas from the camera's viewpoint. The brush is then rendered with each of its texture maps and with the depth testing function set to farthest Z (instead of the usual, nearest Z). The resulting footprints are then read back from the frame buffer for use in blending. The top row of Figure 4.3 shows a set of these footprint images.

When the brush is not a connected mesh, as is the case with the bristle strip representation (Chapter 3), a slightly amended procedure is required. For reasons that will be explained in Section 4.5.3, in order to update the brush textures of a bristle strip brush in the end, a fourth footprint image—one with texture coordinates—is needed. The bottom row of Figure 4.3 shows the set of four footprint images from a non-mesh fan brush.

Before rendering each of the footprints, the framebuffer is initialized with an alpha value of 0. The brush is then rendered with an alpha of 255. The result is that only fragments that are part of the footprint have nonzero alpha. This alpha mask delineates the actual shape of the footprint and is used later when updating the brush textures (Section 4.5.3).

Though dAb only supports a planar canvas currently, to handle painting on a non-planar canvas (i.e. a 3D object), a similar process could be followed by introducing a temporary mapping plane in between the brush and canvas that approximates the regions of contact Ω_{BC} and Ω_{CB}. An additional step is then required to remap the final blended footprint back to canvas texture space, just as is currently done to remap the canvas footprint back to brush texture space (Section 4.5.3).

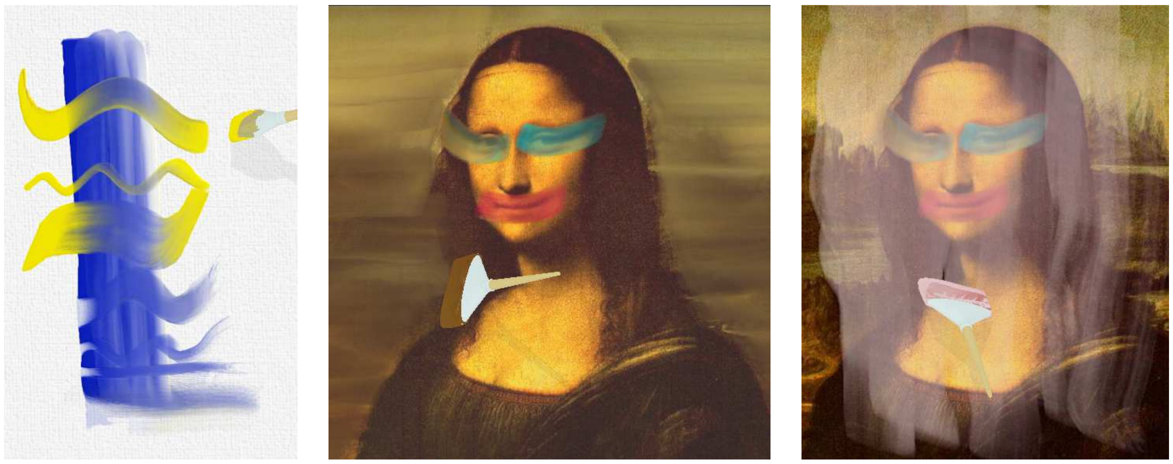


Figure 4.4: *(left) Bi-directional paint transfer is demonstrated by dragging three brushes loaded with yellow paint and four clean, dry brushes through wet blue paint. (center) After making the Mona Lisa's paint wet again for the first time in centuries, she has been given a make-over and the background defocused for dramatic effect using bi-directional transfer to smear the paint. (right) "Mona Lisa Through a Veil", created by glazing over a dry Mona Lisa with with translucent white paint.*

4.5.2 Bi-directional Paint Transfer and Blending

As a stroke is painted, paint from the brush is deposited onto the canvas, but at the same time the brush picks up paint off the canvas. This bi-directional transfer operation enables paint-like mixing effects as shown in Figure 4.4.

This section describes the blending operation (Algorithm 1, line 5; Algorithm 2, line 6) that serves as dAb's bi-directional transfer mechanism. The key observation is that once the 2D footprint textures have been created (previous section), one can blend the canvas paint with the footprint paint as if the 2D footprints *are* the brush.

At this point, since the brush is now a 2D raster, the blending could be carried out essentially like it is in any existing paint program, if so desired. dAb currently implements one blending function, one that simulates the smudging and smearing of real painting, but essentially any of the Shoup model blending modes (Shoup, 2001), or modes added by more recent descendants of the Shoup model, could be used instead. The simple color deposition model used by dAb is in fact itself probably not tremendously different from marking functions used in existing paint programs. The main differences, however, are that 1) dAb allows modification of the color on the brush during mark-making, not just the color of the canvas, and 2) the actual shape of the footprint is derived from contact with a deforming 3D brush geometry. These modifications make a vast difference in the degree to which the painting program begins to feel like actual painting.

In dAb, blending only performs an exchange of paint between the surface layers of brush (footprint) and canvas, as shown in Figure 4.2. The amount of volume transferred between surface layers is dependent on the volume of paint within each layer. The volume leaving each surface, V_l, is computed from the initial volume, V_i, and transfer rate, R, over the elapsed time, T, by the equation, $V_l = V_i + T \cdot R$. The resulting paint color, C_{new}, is computed by the weighted portions of remaining volume and color, $V_r = V_i - V_l$ and C_i, and incoming volume and color from the other surface, V_l' and C_i':

$$C_{new} = (V_r \cdot C_i + V_l' \cdot C_i')/(V_r + V_l')$$

In other words, the new color at each point is a weighted sum of the colors on brush and canvas, with the respective volumes of paint serving as weights. The same equation is used to update both the footprint paint and the canvas paint.

This linear weighted additive color mixing formula is easy to work with, and gives predictable and generally expected results. For example mixing black with any color makes a darker shade of that color, as expected. However, mixing equal parts of something like saturated yellow, (255,255,0) in RGB, with saturated cyan, (0,255,255), does not result in a saturated tint of green as would be expected from real-world pigments, but rather a desaturated green (128,255,128). A purely subtractive model as implemented in (Cockshott, 1991) or Corel Painter's mixing palette (Painter 8, 2003; Zimmer, 1994) would mix the two to get something closer to (0,255,0). The yellow subtracts blue and cyan subtracts red, leaving only green. This would likely be more intuitive for painters to work with.

In truth, however, the difference between RGB and real paint is not simply a matter of additive versus subtractive color spaces. The color of a pigmented material like paint is a complex combination of subsurface scattering and absorption, and accurately modeling the color of pigmented mixtures is still an active area of research. The dAb model, however, strives for maximal simplicity, and so, like most existing painting programs, uses additive RGB mixing. The next two paint models I will present use a more physically-based rendering scheme based on the Kubelka-Munk equations (Kubelka & Munk, 1931; Kubelka, 1948; Kubelka, 1954; Haase & Meyer, 1992). This rendering method is the subject of Chapter 7, and I will discuss color mixing issues further there.

4.5.3 Updating Brush Textures

After determining how paint from the brush and canvas blend within the footprint, dAb maps the resulting blended footprint paint back to the brush to update the brush's textures (line 7 of both algorithms). This also corresponds to the computation of $\mathcal{A}_C \circ \mathcal{T}_{BC}$ described in Section 4.2. For a brush surface represented as a connected polygonal mesh, there is a simple way to map the blended paint in the canvas footprint back to the brush's texture space. The key idea is to rasterize the brush mesh, but with geometric vertex coordinates and texture coordinates swapped. More specifically, one rasterizes using the brush texture itself as a destination buffer, and the buffers containing the canvas space footprints as source texture maps. In terms of the rendering commands issued, that means for each vertex in the brush mesh, use that vertex's *texture* coordinate as a *geometric* coordinate, and use its projected 2D *geometric* coordinate in the footprint as the *texture* coordinate. As long as texture space is diffeomorphic to the geometric space, this generates the correct result for mapping the canvas footprint back to brush texture space, and the rasterization hardware's bilinear interpolation performs the desired interpolation. The rendering should be performed with OpenGL's alpha test enabled so that portions of the footprint buffer that have alpha=0 do not modify the brush texture.

When the brush is *not* a connected mesh, as is the case with the bristle strip representation (Chapter 3), a different procedure is called for. The mapping from canvas texture coordinates to brush texture coordinates is computed explicitly by rasterizing a fourth brush footprint using texture coordinates as color (see Figure 4.3). The resulting texture coordinate image gives the interpolated brush texture coordinate closest to each canvas texture coordinate. Then the blended paint is mapped from the canvas footprint back to brush space in software, using the texture coordinate footprint as an index texture.

4.6 Rendering

After a stroke has been painted, all tiles through which the stroke passed are marked dirty. When time permits, dAb lazily updates the the dirty tiles of the cached rendering used to display the canvas to the user. Updating a tile involves optical composition of the surface layer over the deep layer and embossing to give the canvas a 3D appearance. This section describes those two operations.

4.6.1 Optical Composition

To allow for layering effects like glazing (illustrated in Fig. 4.4), the wet and dry layers of the canvas are composited together, allowing the dry layer to partially show through the wet layer. The volume of the wet layer, V_w, is multiplied by the optical thickness,

O_t, of the paint, and then used for alpha blending the wet and dry layer colors, C_w and C_d. The displayed canvas color is computed as follows:

$$C_{\text{displayed}} = \alpha \cdot C_w + (1 - \alpha) \cdot C_d; \quad \alpha = \min(V_w \cdot O_t, 1). \tag{4.1}$$

4.6.2 Embossing

When rendering the canvas dAb performs bump mapping using the height field (paint volume) to give the canvas a 3D, impasto-like appearance. This bump mapping effect is computed using a simple 3×3 embossing filter kernel applied to the paint height. The embossing operation effectively encodes a fixed light position above and to the right of the canvas. The filter kernel is:

$$\begin{bmatrix} 0 & -1 & -2 \\ 1 & 0 & -1 \\ 2 & 1 & 0 \end{bmatrix}.$$

This technique is fast and efficient, and appropriate for use in interactive graphics.

After evaluating the kernel at each canvas pixel, dAb adds the resulting positive or negative scalar, multiplied by a user controllable "bumpiness" parameter, to all three color channels and clamps the result to yield the final rendered color for the canvas.

In addition to computing the embossing effect due to the wet layer's volume, dAb also adds an embossing contribution from the dry layer's height. To make this operation efficient, dAb stores the post-filtering embossing values for the dry layer in a channel of the canvas' attribute texture (Table 4.1). These values need only be recomputed when the dry layer changes, which only happens during a drying operation. Since the embossing is a linear operation, when drying, it is possible to compute the embossing

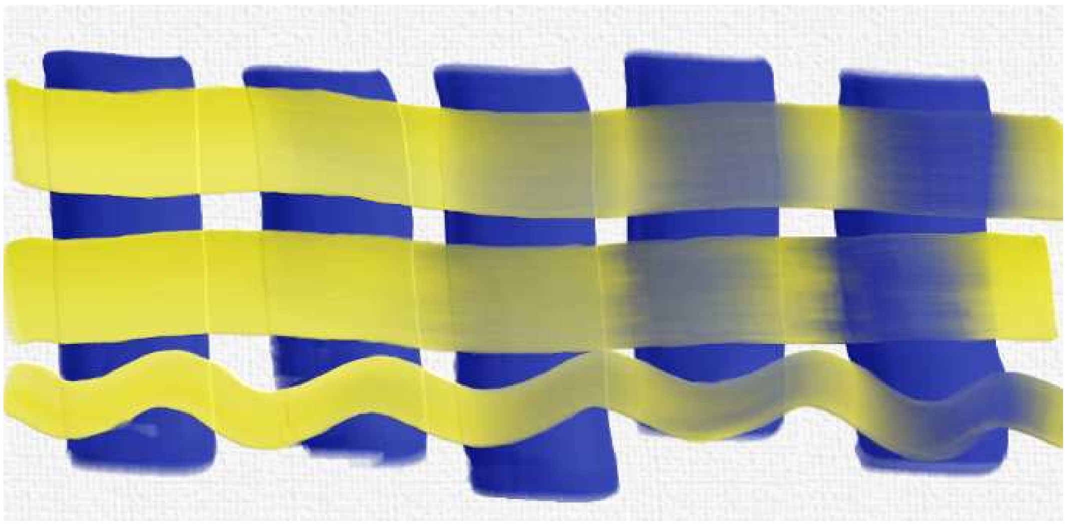


Figure 4.5: *Partial drying is demonstrated by the yellow paint that has been painted over blue color stripes of 0%, 25%, 50%, 75%, and 100% wetness (left to right).*

due to the just the incoming paint volume and add this onto the stored embossing value per pixel.

To compute the lighting, dAb adds the dry layer's embossing contribution to the wet layer's contribution to get the total lighting effect. The dry layer embossing value is stored in as an 8-bit signed format giving it a range of $[-127, 128]$.

4.7 Drying the Canvas

The dAb paint model also supports controlled drying as shown in Fig. 4.5. Partial drying is accomplished by moving a fraction of paint from the completely wet canvas surface layer to the completely dry deep layer.

When paint volume is transferred from the wet layer to the dry layer during drying, the composited color of the paint should not change. The color of real oil paint changes very slightly as it dries, but the *lack* of a dramatic shift in color is one of its more desirable properties (Mayer, 1991). To enforce color constancy, one can use the optical blending function (Equation 4.1) and solve for the new dry layer color. Given a volume,

$\delta\alpha$, to be removed from the wet layer, set the pre-dry and post-dry composite colors equal and solve for the new dry layer color, C'_d. The result is

$$C'_d = \frac{\alpha \cdot C_w + (1-\alpha) \cdot C_d - \alpha' C_w}{(1-\alpha')}; \quad \alpha' = \alpha - \delta\alpha.$$

4.8 Relative Height Field

In the attribute texture of the canvas' dry layer (Table 4.1), dAb also uses one 8-bit channel to store a relative height field that allows a practically unlimited amount of volume to be accumulated over the course of multiple drying operations. The relative height field is not used in the runtime system, but is saved with the canvas so that the full height field of the canvas can be reconstructed offline if desired (for creating a printout on a 3D printer, for instance.)

Let $h(i,j)$ be the value stored in the relative height field for column i and row j. Let $\text{absheight}(i,j)$ be the absolute height of that same row and column. The relative height field is then recursively defined as follows:

$$\begin{aligned}
\text{absheight}(0,0) &:= 0 \\
\text{absheight}(0,j) &:= h(0,j) + h(0,j-1) \\
\text{absheight}(i,j) &:= h(i,j) + h(i-1,j).
\end{aligned}$$

This is a simple and efficient means of encoding the height with reduced memory overhead compared to a full 32-bit per pixel representation. It does, however, impose a limitation on the maximum height difference between adjacent pixels.

4.9 The Palette

The dAb system was the first to introduce the virtual palette as an interface for color mixing and performing complex brush loading. Rather than requiring the user to select color from a fixed list, or pick colors by number, the user simply picks pure colors from paint wells and mixes them together to get the desired tones. The overall interface, including the palette, will be discussed more in Chapter 8, but there are several implementation issues that are more appropriately discussed here.

The dAb palette is implemented as a special type of canvas. It uses the two-layer model just like the canvas, however it has a few special properties. First it supports three different modified brush transfer modes: normal palette mode, paintwell mode, and brush cleaner mode. Which region of the palette uses which mode is determined by a region code stored in the palette's attribute texture (Table 4.1).

First, the normal palette mode: this mode is different from the standard canvas in that contact with the palette causes the brush's deep reservoir layer to be refilled. This does not happen with the normal transfer algorithm used on the canvas. The palette also never loses paint volume, so that the brush can be refilled as often as desired with a particular user-created mixture on the palette. Next, regions of the palette in paintwell mode have permanent color. The brush gains paint when in contact with a paintwell, but the the paintwell never loses paint or has its color modified. Finally, the brush cleaner regions: these regions are also non-modifiable, like the paintwells, but instead of refilling the brush, they empty out the brush's surface and deep layer volumes, cleaning the brush so it can be used on the canvas with a dry-brush technique.

Together these three modes enable the user a wide range of options for how load their brush with just the blend of color desired.

dAb does not perform any embossing on the palette, since it is meant for mixing, not painting pictures. It does, however, allow the palette to have a non-rectangular

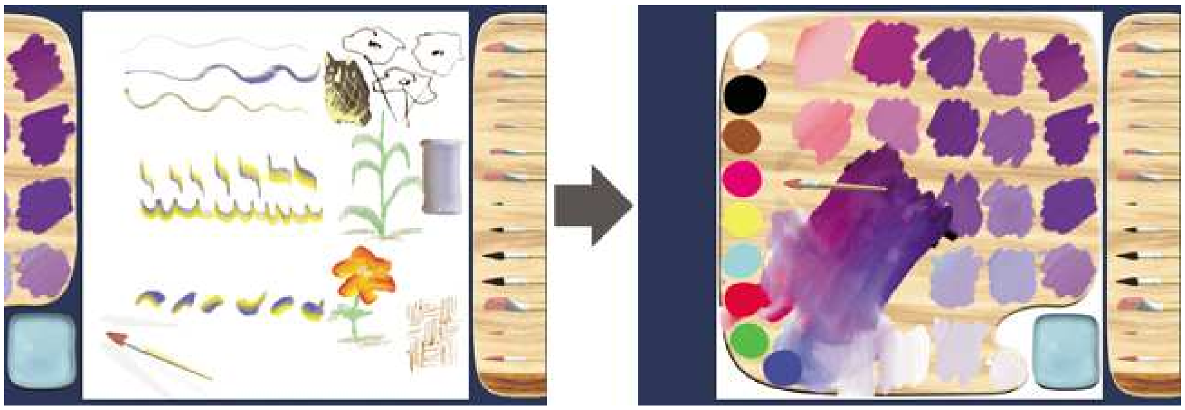


Figure 4.6: *The dAb virtual palette. (left) Tucked away to the left of the canvas. (right) After sliding into full view for color mixing. Paintwells appear along the left edge, and the brush cleaner in the lower right. The remainder of the area is "normal palette".*

shape through the use of a shape mask in the palette attribute texture (Table 4.1). The shape mask is used as an alpha channel when rendering the palette, which enables it to take on a more aesthetic, rounded appearance, more like a real artist's palette (see Figure 4.6).

4.10 Implementation and Results

The dAb paint model was implemented in C++ using the OpenGL graphics API. dAb divides the work between the CPU and GPU, generating footprints using graphics hardware, but performing the actual blending and compositing operations in software. For efficient execution, all of the CPU operations were implemented using fixed point arithmetic. As a consequence of this division of labor, the dAb paint algorithm will run on any graphics hardware that supports the basic OpenGL 1.1 specification.

I tested the performance of the two stroking algorithms on several machines, to demonstrate the effectiveness of the algorithm over a wide range of system capabilities. The results are summarized in Table 4.2.

Performance (10^6 Texels/sec)

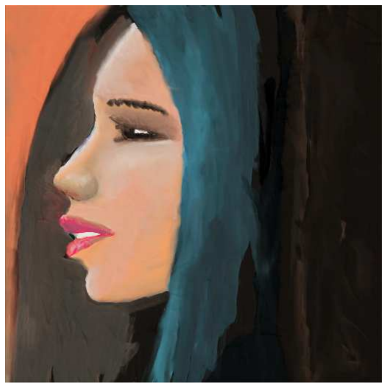
	Path 1			**Path 2**		
	System[1]	System[2]	System[3]	System[1]	System[2]	System[3]
Algorithm 1	2.39	5.84	8.39	1.62	5.22	7.58
Algorithm 2	0.97	1.17	2.93	0.96	1.47	3.39

Table 4.2: *Measured performance of the dAb paint stroking algorithms, in millions of canvas texels updated per second, for two example brush paths. The additional readback from graphics memory required by Algorithm 2 makes it significantly slower in every case, as expected.* System[1] *is a 1.4GHz Intel Pentium M laptop with 512MB RAM, and an NVIDIA GeForce FX Go5200 with 32MB of RAM on an AGP 4x bus.* System[2] *is a 2.53GHz Intel Pentium 4 with 1GB RAM, and an NVIDIA GeForce FX 5900 Ultra with 256MB of RAM on an AGP 4x bus.* System[3] *is a 3.40GHz Intel Pentium 4 with 2GB RAM, and an NVIDIA GeForce FX 6800 Ultra with 256MB of RAM on an AGP 8x bus.*

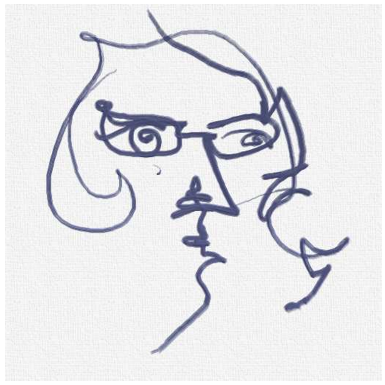
Many artists have tried dAb, and some of the images they created can be seen in Figure 4.7. The paintings demonstrated in the last chapter were also created with the dAb paint model. Many visiting groups of school children have also tried it and always seem to enjoy themselves. dAb was demonstrated to a large audience at the Creative Applications Lab in SIGGRAPH 2001; it has also been demonstrated at Pixar Animation Studios; and in 2002 it was shown at Intel's annual research fair in Beaverton, OR where it was voted "best project". Since 2004, SensAble Technologies has been using dAb as a demonstration of their haptic technology at trade shows; and since its creation in 2001 it has been demonstrated to hundreds of visitors to the computer science department of UNC Chapel Hill.

4.11 Limitations

The dAb paint model is simple, fast and enjoyable to paint with; however, the model has its limitations. The primary issue with dAb as a thick impasto model is that thickness is only accumulated onto the canvas' deep layer after drying, and it is only a visual thickness, created by embossing, so it cannot be pushed around or manipulated

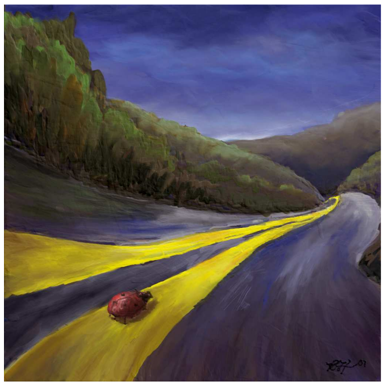
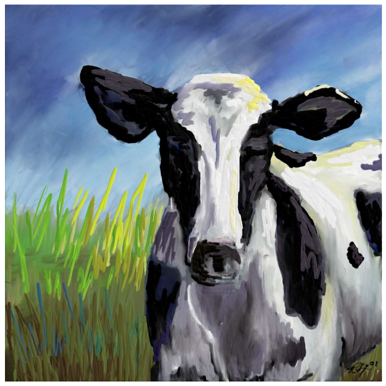


Andrei State *Eriko Baxter* *Lauren Adams*



Lauren Adams *Rebecca Holmberg* *Rebecca Holmberg*

Rebecca Holmberg *Rebecca Holmberg* *William Baxter*

Sarah Hoff *Rebecca Holmberg* *Rebecca Holmberg*

Figure 4.7: *Paintings created by various artists using dAb. The paintings shown in the previous chapter were also created with the dAb paint model.*

by the painter, nor does the brush react any differently to it. This limitation was the main motivation for developing the paint models described in the next two chapters.

The color model is another issue, as the colors do not mix or glaze like real paint colors do. A much improved rendering scheme is the subject of Chapter 7.

4.12 Summary

In this chapter I have presented the dAb paint model. The main contributions of this chapter are:

- A general mathematical description of attribute mapping operations for computing footprints.
- Specific bi-directional footprint computation algorithms, which leverage rasterization hardware, for both connected, closed mesh objects and for meshes with holes or gaps. These algorithms form the connection between the 3D brush models of Chapter 3 and the paint models of this and Chapter 6.
- Two algorithms for painting smooth, connected strokes, one offering better performance, and the other better quality.
- A simple, interactive 2-layer paint model that captures many of the characteristics of paint using a minimal amount of memory and computation.

The next two chapters will present paint models that improve upon dAb in several ways.

Chapter 5

Stokes Paint: 3D Volumetric Paint

Painting is easy when you don't know how, but very difficult when you do.
— Edgar Degas

This chapter presents Stokes Paint, the first painting system to be based on a full 3D volumetric fluid model for viscous paint. The fluid behavior of the paint is based on the 3D Stokes equations. In contrast to the dAb model, this Stokes Paint model enables physically-based interaction with the paint and multiple active, wet layers. The result is a system that, while much more computationally intensive, allows for realistic pushing, squishing, and scraping of paint to create paintings in styles similar to impasto[1] or encaustic[2]. For rendering, Stokes Paint incorporates the real-time Kubelka-Munk reflectance model that will be described in Chapter 7.

As a paint model, Stokes Paint supports similar operations to the dAb model: blending, bidirectional paint transfer, controlled drying, glazing and impasto. The main features of the Stokes Paint model that are responsible for the improved material characteristics listed above are:

- A true volumetric 3D fluid representation.

[1]Impasto is a painting style in which paint is applied thickly with little additional medium.

[2]In encaustic painting, molten wax serves as the painting medium.

- Physically-based fluid motion derived from the Stokes differential equations.
- Multiple layers of paint all active (wet) simultaneously.

I have integrated the Stokes Paint paint model into the painting system framework developed originally for the dAb system, to demonstrate the interactive capabilities of the viscous fluid paint model. Although at this time the system aspects of Stokes Paint are less developed than the other two paint models (dAb and IMPaSTo, to be described in the next chapter), the integration nonetheless serves as a powerful proof-of-concept demonstration of the potential for fully 3D digital painting based on fluid simulation.

This chapter consists of two main parts. The first part discusses the details of the numerical method I use to solve for the fluid behavior of the paint. The second part of the chapter is specifically about the issues involved in integrating this fluid model into a proof-of-concept painting system. Though much of the previous work related to computer based painting has been discussed already in Chapter 2, the area of fluid simulation was not covered there. I will review the fluid related research literature briefly before proceeding to the primary topics of this chapter.

5.1 Previous Work in Fluid Simulation

Many researchers have investigated simulations of fluid for computer graphics. I present a brief summary of the most relevant related work below. In this chapter we are interested particularly in fluid simulation techniques that are 1) interactive, 2) suitable for high viscosity, 3) can handle a free surface boundary, and 4) describe the entire volumetric flow rather than just the surface motion. It will be seen that none of the previous work addresses all of these concerns adequately for viscous painting.

Kass and Miller (Kass & Miller, 1990) used the linearized shallow-water equations to simulate surface waves. The method is fast, stable and interactive, but cannot handle viscous flow, and only simulates the surface height. Subsurface flow is not computed. O'Brien and Hodgins (O'Brien & Hodgins, 1995) combined a particle system with shallow-water equations to simulate splashing of low viscosity fluid. (Hinsinger et al., 2002) presented a method for the interactive animation of ocean waves.

Chen and Lobo (Chen & Lobo, 1995) used the 2D Navier-Stokes equations, taking pressure resulting from the solution to be proportional to height to get the third dimension. The method is interactive, though using pressure as height is a fairly gross approximation of the behavior of 3D fluid. Also, since the method is fundamentally 2D, the subsurface flow and mixing are unknown.

Desbrun and Cani presented an interactive particle-based technique for simulating a range of materials from highly deformable solids to fluid-like substances (Desbrun & Cani, 1996), and Muller, et al. also proposed an interactive particle-based method using Smoothed Particle Hydrodynamics (SPH) to simulate fluids with free surfaces (Muller et al., 2003). The drawback to both methods for viscous paint simulation is that they perform poorly when the particles are spread too thinly, which is one of the more important operations necessary in a paint simulation.

Foster and Metaxis (Foster & Metaxas, 1996) used an explicit marker-and-cell (MAC) method based on (Harlow & Welch, 1965) to simulate low viscosity free-surface liquid. Being an explicit method, it is subject to the CFL and viscosity timestep restrictions ($\Delta t < O(\Delta x)$ and $\Delta t < O(\Delta x^2)$, respectively), making it unsuitable for use in interactive applications.

Curtis, *et al.* (Curtis et al., 1997) used a form of the shallow water equations in their watercolor simulation. Their explicit formulation is subject to timestep restrictions and is inappropriate for very viscous or very thick layers of fluid.

Stam (Stam, 1999) introduced the first unconditionally stable solver for the Navier-Stokes equations to the graphics community. The solver's use of implicit backwards-

Euler integration for viscosity allows for stable treatment of high viscosity fluids, but the method does not address the complications or stability issues introduced by the presence of a free surface boundary condition.

(Fedkiw et al., 2001) used incompressible, inviscid Euler equations to simulate smoke. Recently, (Foster & Fedkiw, 2001; Enright et al., 2002) have presented convincingly accurate particle level set methods for low-viscosity free surface flow, but these methods are quite computationally intensive, requiring minutes per frame for simulation.

Carlson, *et al.* (Carlson et al., 2002) presented simulations of melting and flowing of high-viscosity fluids based on the MAC method. While their method treats viscosity implicitly, advection is still performed explicitly, making it subject to the CFL timestep restriction. Also their method for handling the free surface boundary conditions is not explicitly discussed and likely subject to a timestep restriction as well, making interactivity difficult to achieve.

Goktekin *et al.* (Goktekin et al., 2004) present an extension to (Enright et al., 2002) that allows for simulation of visco-elastic and viscous fluids. Their results are impressive, but like other particle level-set methods, reported simulation times are as high as one minute per simulation timestep, making the method currently far from practical in an interactive setting.

The most closely related work, at least in terms of goals, is that of (Cockshott et al., 1992), already mentioned in Chapter 2. He presents a cellular-automata model for the dripping and flowing behavior of runny liquid paint on a canvas. The method was not interactive on the 2MHz processors available at the time, though it likely would be today with processors that are 1000 times faster. The fluid model Cockshott presents, however, recreates the behavior of a fairly low viscosity fluid. Furthermore he does not propose any approach for how a brush should interact with this paint model to create a painting system.

5.2 Governing Equations

Paint is a complex material, composed of millions of tiny powdered pigment particles[3] suspended in a small amount of medium such as linseed oil. As a consequence of their composition, paints can have both liquid- and solid-like properties. Oil paint, for example, as it comes straight out of the tube, does not begin to flow until its internal yield stress has been exceeded by shear stress. When squeezed out, it initially maintains a rigid cylindrical shape, despite the action of gravitational forces. After enough of the material emerges, however, gravitational forces create enough shear stress to exceed the yield stress, and it begins to deform in a fluid-like manner.

From that simple example it is clear that paint is not a typical fluid. In fact, it is a member of the general class of materials known as a non-Newtonian fluids. A non-Newtonian fluid is one in which the viscosity varies depending upon shear rate. The behavior described above is sometimes known as "viscoplasticity". Other viscoplastic fluids include drilling mud, toothpaste, and blood (Subramanian, 2003). Another non-Newtonian property of paint is "shear-thinning", that is, its viscosity decreases as shear rate increases. For painting this is a desirable property in that the paint will flow smoothly when sheared by a brush, but after the shear stress is removed, it returns to a higher viscosity so it no longer flows easily (Subramanian, 2003).

The momentum equation for a general non-Newtonian fluid can be written

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \frac{1}{\rho}\nabla \cdot \sigma + \mathbf{F} \tag{5.1}$$

[3]The size of the pigment particles used in paints range from 0.1-50μm, with 5μm being the mean(Callet, 1996).

where $\mathbf{u}$ is the fluid velocity field, p is the pressure field, ρ is the density, and $\mathbf{F}$ is an external force per unit mass such as supplied by gravity. The stress tensor, σ, determines the relationship between stress and strain in the fluid. In a Newtonian fluid the stress and strain are linearly related, but in a non-Newtonian fluid this relationship is non-linear, and in particular the shearing components of the stress tensor become dependent on the rate of shear. Thus the effective viscosity of a non-Newtonian fluid is also a function of shear rate.

Developing a model that accurately and fully models the viscoplastic, non-Newtonian nature of paint is a daunting challenge, even more so if the model must also be interactive. Both (Carlson et al., 2002) and (Goktekin et al., 2004) present simulation techniques for handling particular types of fluids with some solid-like behaviors, the former by modeling solids as very high viscosity fluids, the latter by using a viscoelastic model of the material. Both techniques are able to generate high-quality animations offline, but for painting, an interactive technique is needed.

Instead of trying to explicitly model the non-Newtonian nature of the paint, I have created a three-regime model. The first regime is simply a solid-like regime, in which the paint is static. This is the case for undisturbed paint applied to the canvas. The second is the high-viscosity regime, which is the subject of the majority of this section. The third regime is the shear-thinned regime, which is active in the immediate vicinity of the brush and which represents the reduced-viscosity state of the paint after the brush applies shearing forces. This is implemented using a filtering operation discussed in Section 5.4.4.

Stokes Paint simulates the high-viscosity fluid regime of paint using the 3D incompressible unsteady Stokes equations:

$$\frac{\partial \mathbf{u}}{\partial t} = \nu \nabla^2 \mathbf{u} - \nabla p \tag{5.2}$$

where ν, the kinematic viscosity, is now a constant. For convenience, constant density is assumed, since most familiar viscous paints are approximately homogeneous. Equation 5.2 is coupled with the equation of continuity,

$$\nabla \cdot \mathbf{u} = 0, \tag{5.3}$$

which enforces incompressibility and the conservation of mass. The Stokes equation is a simplification of Navier-Stokes applicable for highly viscous (low Reynold's number) fluids. The simplification comes primarily from dropping the advection term, $(\mathbf{u} \cdot \nabla)\mathbf{u}$, from the full Navier-Stokes equations, and from the assumption of constant viscosity.

As mentioned, the Stokes equations are a reasonable approximation of Navier-Stokes for low Reynolds number flows. The Reynolds number is a dimensionless constant that roughly describes the ratio of inertial forces to viscous forces in in fluid. For Stokes to apply, the Reynolds number should be less than 1. One can calculate the Reynolds number of a typical painting scenario to verify that this is the case. The formula for the Reynolds number is

$$Re = \frac{VL}{\nu},$$

where V is a characteristic velocity, L is a characteristic length, and ν is the kinematic viscosity of the fluid. The viscosity of typical oil paints directly out of the tube is over 10,000 centiStokes (cSt) [4], though as noted, the actual value can be higher depending on the rate of shear. One centiStoke is $10^{-6}\text{m}^2/\text{s}$. A swift stroking motion may move at about 0.3 m/s, and the size of a typical paint brush head is on the order of 0.01m.

[4] It is difficult to obtain much information about the fluid dynamic properties of artistic paints; however, viscosities of similar materials like toothpaste, or molasses can be found more readily. For example (Carlson et al., 2002) lists 10,000 cSt as the viscosity of toothpaste, and various other sources list viscosities for molasses as about the same.

The result is

$$Re = \frac{0.3m/s \cdot 0.01m}{10,000 \cdot 10^{-6} m^2/s} = 0.3,$$

which is is a fairly conservative estimate of what will probably be the maximum Reynolds number. Brushes are frequently smaller, motion slower, and the viscosity higher, meaning lower Reynolds numbers. So one can conclude that the Stokes approximation is reasonable for the purposes of painting.

5.3 Numerical Method

The Stokes Paint solver uses a staggered 3D grid as in (Harlow & Welch, 1965; Foster & Metaxas, 1996; Griebel et al., 1990) and others, with the vector components such as velocity stored on cell edges, and scalar quantities (including color channels) stored at cell centers.

The Stokes Paint model uses a hybrid phase-field/volume-of-fluid technique to track the free surface and is stable for any size of timestep. The numerical solver I have developed has the following characteristics:

- Stable implicit viscosity step with semi-Lagrangian update of surface and color, and free surface boundary treatment;
- Hybrid linear system solver combining incomplete Cholesky preconditioned conjugate gradient method with successive over-relaxation;
- Real-time performance suitable for interactive applications.

The numerical method used to solve the fluid flow equations is a fractional step, Chorin projection method similar to some previous methods, e.g. (Foster & Metaxas, 1996; Stam, 1999; Carlson et al., 2002). A provisional velocity field, $\mathbf{u}^*$, is first computed to capture the effect of the viscous term, $\nu\nabla^2\mathbf{u}$. This step uses a stable backwards-Euler integration step. A Poisson problem to find a pressure field, p, is then solved for to make $\mathbf{u}^*$ discretely satisfy the incompressibility constraint, Equation 5.3. Once obtained, the new pressure, p, is used to compute the final divergence-free velocity field, $\mathbf{u}$.

The above three-step temporal discretization scheme can be written succinctly as follows:

$$\mathbf{u}^* = \mathbf{u}^n + \Delta t[\nu\nabla^2\mathbf{u}^*] \tag{5.4}$$

$$\nabla^2 p = \nabla \cdot \mathbf{u}^*/\Delta t \tag{5.5}$$

$$\mathbf{u}^{n+1} := \mathbf{u}^* + \Delta t\nabla p \tag{5.6}$$

where the superscripts on $\mathbf{u}$ refer to the time step at which the variables are to be evaluated. Stokes Paint uses boundary conditions to model the forces applied by the artist through the brush.

To model and track the evolution of the free surface—the interface between the fluid and air—Stokes Paint uses a hybrid method which is similar to phase-field methods or the volume-of-fluid method of (Hirt & Nichols, 1981) in that every cell in the computational domain is assigned a scalar value between 0 and 1 denoting the phase of the contents, either air or fluid. For the purpose of placing boundary conditions on the simulation, a cell is treated as fluid if its phase value is greater than one half. The precise location of the surface is taken to be the phase = 0.5 isosurface, though this is used only for rendering. The method for extracting the isosurface is discussed in Section 5.3.5. Unlike some previous free surface methods, each step of this numerical method is stable, allowing the solver to take large time steps and maintain interactivity.

5.3.1 Viscosity

Stokes Paint solves for the effect of viscosity using an implicit Euler update (Equation 5.4), which is unconditionally stable (Stam, 1999; Carlson et al., 2002). The spatial discretization of Equation 5.4 leads to a system of equations, $K\mathbf{u}^* = \mathbf{u}^n$,where $K = \mathbf{I} - \nu\Delta t\nabla^2_{\mathrm{D}}$ and ∇^2_{D} is the standard 7-point (3D) Laplacian stencil in matrix form. The system is actually three independent systems of equations, one for each velocity component, u^*, v^*, and w^*. Expanding the compact matrix notation above out into its constituent linear equations, the system of equations for the u^* component is:

$$\begin{aligned} K_c u^*_{i,j,k} + K_x(u^*_{i-1,j,k} + u^*_{i+1,j,k}) & \\ +K_y(u^*_{i,j-1,k} + u^*_{i,j+1,k}) & \\ +K_z(u^*_{i,j,k-1} + u^*_{i,j,k+1}) &= u^n_{i,j,k} \end{aligned} \tag{5.7}$$

where

$$\begin{aligned} K_c &= 1 + 2\nu\Delta t(1/\Delta y^2 + 1/\Delta x^2 + 1/\Delta z^2) \\ K_x &= -\nu\Delta t/\Delta x^2 \\ K_y &= -\nu\Delta t/\Delta y^2 \\ K_z &= -\nu\Delta t/\Delta z^2 \end{aligned}$$

Written as a matrix, K is a $D^3 \times D^3$ matrix, where D is the number of samples on each dimension of the 3D grid, but the matrix is very sparse, containing only $O(D^3)$ non-zero entries, making it amenable to solution with the conjugate gradient method. I use the conjugate gradient method with an incomplete Cholesky preconditioner. Pseudo-code algorithms for the conjugate gradient method as well as the preconditioner can be found in (Golub & Van Loan, 1983). An excellent description of the conjugate gradient method is available in (Shewchuk, 1994).

The sparse matrix multiplies required by the conjugate gradient solver can be implemented simply by applying the matrix stencil to the grid, i.e. evaluating the right hand side of (5.7), at each (i, j, k) on the domain. For example:

```
for k = 0 to DEPTH
 for j = 0 to HEIGHT
   for i = 0 to WIDTH
    if (i,j,k)  in  domain
    then uout(i,j,k) := K_c * uin(i,j,k)
       + K_x (uin(i-1,j,k)+uin(i+1,j,k))
       + K_y (uin(i,j-1,k)+uin(i,j+1,k))
       + K_z (uin(i,j,k-1)+uin(i,j,k+1));
```

would compute the product $K * u_{in}$. The check in line 4 to see if the grid cell is in the domain is used when handling domains with irregular geometry, as is the case with a rough-surfaced canvas. For the implicit viscosity solver, this check is true if the cell in question is a fluid cell.

5.3.2 Pressure Solver

Given the tentative velocity field, $\mathbf{u}^*$, the next step is to find a pressure field such that the divergence of $\mathbf{u}^* - \Delta t \nabla p$ is near zero by solving the Poisson problem (Equation 5.5). For low viscosity flows, inertial forces dominate (i.e., advection) so there is a high degree of temporal coherence in the velocity field. Consequently, a small number of iterations of successive over-relaxation (SOR) per timestep is sufficient to yield realistic-looking results (Foster & Metaxas, 1996). However, in very viscous flow, momentum spreads out quickly, creating large velocity gradients and low temporal coherence. Thus, it is necessary to use more solver iterations to enforce incompressibility as viscosity increases. After experimenting with several different schemes, I have found a particularly effective approach to be a combination of both conjugate gradient (CG) and SOR. My SOR solver steps are identical to those in (Foster & Metaxas, 1996; Griebel et al., 1990). I use between 10-15 iterations of CG with an incomplete Cholesky preconditioner, followed by 3 or 4 iterations of SOR. The residual after applying CG tends to have a fair amount of high frequency content since CG is a "rougher", in the terminology of (Shewchuk, 1994). A few iterations of SOR applied after CG is particularly effective since SOR acts as a "smoother". I show examples of typical residuals after several iterations of CG and SOR in Figure 5.1. In my tests, the CG/SOR combination was quantitatively more effective per CPU second than either technique alone. Comparisons were made by calculating convergence ratios for each technique given the same initial conditions and dividing the result by the computational time required. A multigrid solver is another viable option for the solution, though the rules for handling the interpolation and restriction operations in the presence of complex boundaries can be difficult to implement.

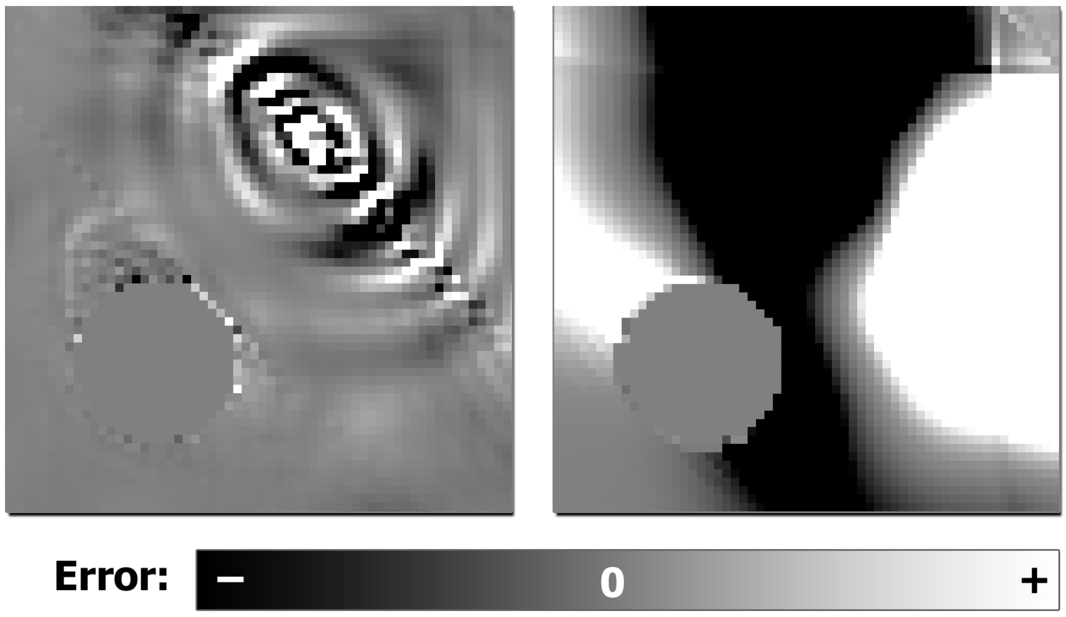


Figure 5.1: *Comparison of error residuals in the Poisson equation for pressure after several iterations of (*left*) conjugate gradients (CG) with incomplete Cholesky preconditioner versus (*right*) SOR. Note the low-frequency nature of the SOR residual compared to that of CG.*

5.3.3 Boundary Conditions

Each stage of the numerical method must be coupled with appropriate boundary conditions. For the diffusion step, the no-slip Dirichlet velocity boundary conditions, $\mathbf{u} = 0$, are used at wall boundaries, and the free velocities on the fluid-air interface are set to discretely satisfy the continuity equation (5.3). The boundary conditions are enforced by setting the value of "ghost cells", which lie just outside the domain. For Dirichlet boundary conditions the ghost values on an edge along the interface are simply set to zero. Values just off the interface are set so that $(u_{ghost} + u_{neighbor})/2 = 0$, as shown in Figure 5.2. For details on implementing these types of boundary conditions (Griebel et al., 1990) is highly recommended.

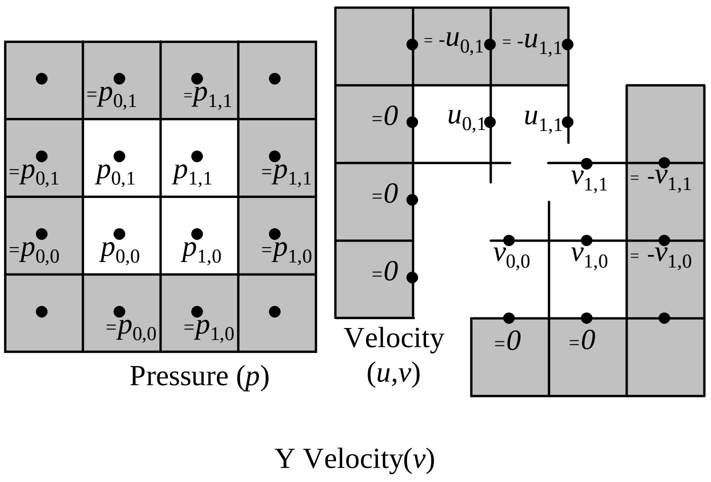


Figure 5.2: *Setting pressure and velocity boundary conditions*

For the pressure Poisson equation, Neumann boundary conditions are required, $\partial p/\partial n = 0$, where n is the boundary normal. These are implemented by copying the pressure value just inside the domain to the ghost cell just outside, before every CG or SOR iteration. Thus on the face of a boundary cell in the positive x direction we have, for example, $(p_{\text{inside}} - p_{\text{ghost}})/\Delta x = 0$, which is the finite difference approximation to the above boundary condition.

5.3.4 Interaction

Rather than adding forcing terms to the Stokes equations to implement interaction with the fluid, one can achieve greater control of the fluid by setting Dirichlet velocity boundary conditions at the fluid surface. The velocities of surface cells adjacent to the virtual brush are simply set to the brush's velocity. This is similar to the approach used for interaction with smoke in (Fedkiw et al., 2001).

5.3.5 Free Surface

Unlike previous approaches, my method for handling the free surface of the fluid is stable even at high viscosity. Stokes Paint represents the surface implicitly as the level set of a fluid phase function, $\phi(i, j, k)$, with the interface defined to lie on the $\phi = 0.5$ isosurface. Insofar as the surface is defined using the level set of an implicit function, this approach is similar to that of (Foster & Fedkiw, 2001; Enright et al., 2002), but unlike those level set approaches, in Stokes Paint the implicit function is not required to be a signed distance function.

The free surface should obey the no-stress conditions, which state that no momentum can be transferred across the interface (Hirt & Shannon, 1968; Nichols & Hirt, 1971; Griebel et al., 1990):

$$\begin{aligned} p - 2\nu(\partial \mathbf{u}_n/\partial n) &= 0 & (5.8)\\ \nu(\partial \mathbf{u}_n/\partial m + \partial \mathbf{u}_m/\partial n) &= 0 & (5.9)\\ \nu(\partial \mathbf{u}_n/\partial b + \partial \mathbf{u}_b/\partial n) &= 0 & (5.10)\end{aligned}$$

where n,m, and b are the surface normal, tangent and binormal, and $\mathbf{u}_q$ is the directional derivative of $\mathbf{u}$ in the q direction, $\nabla\mathbf{u} \cdot q$. These terms have the effect of slowing down surface waves (Hirt & Shannon, 1968; Nichols & Hirt, 1971). This retardation of propagation speed increases with increasing viscosity. At very high viscosity, the free stress forces essentially damp out surface waves instantly. If the free stress conditions are ignored, as in (Foster & Metaxas, 1996), fluids of high viscosity will move unrealistically because the surface cells will tend to retain too much momentum. (Hirt & Shannon, 1968; Nichols & Hirt, 1971) and others incorporated the above free stress terms by solving them for pressure and enforcing that value as a boundary condition on the free surface. However, this approach is unstable for high viscosities.

The source of the instability can be seen by writing out the finite difference approximations for the equation above on surface cell edges. For example, the typical discretization for a surface cell with only one empty cell in the positive x direction is $p_{i,j} = \nu(u_{i,j} - u_{i-1,j})/\Delta x$ (Griebel et al., 1990), where the p value is located at the center of the cell and the u values are on the right and left edges. As viscosity, ν, becomes large, it is clear that any small fluctuation in velocity values will be magnified into a large positive or negative pressure boundary value. The large pressure in turn leads to a large velocity adjustment in the next time step, resulting in an unstable feedback loop.

Fortunately there is a simple solution. Instead of explicitly incorporating the above free-stress equations, or omitting them entirely, I approximate their effect for very viscous fluid by simply zeroing out the surface velocities at the end of every time step. This is a reasonable approximation for the type of paint media I am interested in, and it eliminates the surface instability. With the exception of this important modification, my handling of the free surface boundary conditions is just as in (Foster & Metaxas, 1996; Griebel et al., 1990).

5.3.6 Scalar Advection

After solving for the velocity field $\mathbf{u} = (u, v, w)$, I advance both the phase values and any other material properties, such as color, on the 3D grid using the advection equation for a scalar, s:

$$\frac{\partial s}{\partial t} = -(\mathbf{u} \cdot \nabla)s.$$

I advect using the stable semi-Lagrangian method presented in (Stam, 1999). Specifically, I update the 3D scalar fields by tracing characteristics with an Euler integration step backwards in time:

$$\begin{aligned} \mathbf{x}^* &\equiv \mathbf{x} - \Delta t[\mathbf{u}(\mathbf{x})] \\ f^{n+1}(\mathbf{x}) &= f^n(\mathbf{x}^*). \end{aligned}$$

In general, the source location, $\mathbf{x}^* = (x^*, y^*, z^*)$, will not lie at the center of a cell, so the result is computed using trilinear interpolation of the eight nearest cells. If in backtracking a boundary is crossed, the value of the scalar at the boundary is used.

As a final note, during advection of the color field, a color being advected from a cell with zero ϕ value is taken to have the brush color.

5.3.7 Summary of Method

Here I present a compact summary of all the steps from beginning to end of one time step.

1. Set boundary velocities to zero (viscous stress approximation)
2. Compute $\mathbf{u}^*$ from the implicit diffusion equation
3. Set pressure boundary values according to Neumann boundary condition.
4. Solve pressure Poisson equation

5. Set surface boundary velocities using continuity equation
6. Advect the phase values and color/pigments.
7. Extract surface mesh from phase, and compute surface normals.

5.4 Paint System Integration

The previous section presented the details of the fluid model and the numerical technique used to solve for the flow. In this section I will discuss all the aspects of integrating the fluid model with painting system, including interaction with a 3D brush, drying, and rendering.

5.4.1 Voxelizing the Brush

For depositing paint on the canvas using the brush I have implemented some simple heuristics similar in spirit to those used in dAb, but adapted to the volumetric nature of Stokes Paint.

Stokes Paint starts with a mesh-based brush model as used in dAb, but at the beginning of each time step the brush mesh is rasterized onto the voxel grid and these voxels are tagged as temporary wall boundaries. The velocity at these wall boundaries is set to the brush velocity to cause the nearby paint to move with the velocity of the brush, as was mentioned in Section 5.3.3.

5.4.2 Brush-Canvas Paint Transfer

For simplicity, in the current system the brush is assumed to have only a single color; however, as in dAb, that color is bi-directionally blended with the color of the paint it touches to allow the brush to “pick up” paint from the canvas in addition to just depositing paint. The amount of color change depends on the number and the color of voxels of paint in contact with the brush.

The brush color is used to update the color of any voxels it comes in contact with according to a linear blending equation. Specifically, for each brush voxel the color of the corresponding canvas voxel is updated according to:

$$C_{new} = (1 - stainpow) \cdot C_{old} + stainpow \cdot C_{brush},$$

where C_{new} and C_{old} are the new and old colors of the canvas voxel, C_{brush} is the color of the brush, and *stainpow* is the brush's staining power parameter, between 0 and 1. This color update is applied to both fluid voxels and empty voxels. The empty voxels need to be colored as well because the paint's surface color for rendering is determined by linear interpolation between the two closest voxel colors. The value of *stainpow* is a user-controlled parameter that determines how much influence the brush color has over the canvas paint color.

The brush deposits fluid on the canvas by modifying the phase field values of some canvas voxels. A naïve approach would be to add ϕ to all the voxels the brush touches. Unfortunately the result of such a procedure is a brush that appears to discharge paint into mid-air. This could be the basis for an interesting and novel 3D painting system, but it does not generate physically plausible results. In reality it usually requires some friction to pull viscous paint off of a brush. So instead, the brush should only deposit paint where contact with the canvas or other paint creates the friction necessary to cause paint deposition. After experimenting with several paint deposition rules to try to achieve this behavior, I arrived at the following simple heuristic. For a brush voxel in canvas cell (i, j, k), the ϕ value of the canvas cell is updated using:

$$\begin{aligned}\phi_{\downarrow} &= \max(\phi(i,j,k-1),\ \phi(i,j,k)) \\ \phi_{new}(i,j,k) &= \text{clamp}\left(gain\_rate * \frac{1}{2}(\phi_{\downarrow} + \phi(i,j,k)),\ 0,\ 1\right),\end{aligned}$$

where *gain_rate* is a number slighly greater than unity. With this multiplicative update rule, if the brush is in the middle of empty voxels (all $\phi = 0$), then ϕ stays zero, and no paint is deposited; however, if a brush voxel is above a fluid voxel with a ϕ close to 1, then the canvas voxel is updated with slightly more ϕ than the average of the two, so that the ϕ increases and the canvas voxel is gradually turned into a fluid cell. The solid voxels on the surface of the canvas are initially set to $\phi = 1$ so that paint will also deposit onto bare canvas. The computation of $\phi_{\downarrow}$ ensures that a fluid voxel over an "air bubble" is not drained of paint by this operation.

The brush also picks up color from the canvas. Since Stokes Paint currently only supports a single color for the brush at a time, the new brush color is determined by a summation over all of the canvas voxels the brush occupies, to arrive at an average canvas color weighted by the ϕ in each canvas cell. The equations used are as follows. Let *brushvox* be the set of voxels occupied by the brush:

$$\begin{aligned}
\phi_{tot}^{brush} &= \sum_{brushvox} 1 & (5.11)\\
\phi_{tot}^{canvas} &= \sum_{brushvox} \phi(i,j,k) & (5.12)\\
C_{sum}^{canvas} &= \sum_{brushvox} \phi(i,j,k) * C(i,j,k) & (5.13)\\
dstFactor &= \phi_{tot}^{brush} * (1.0 - brushperm) & (5.14)\\
srcFactor &= \phi_{tot}^{canvas} * brushperm & (5.15)\\
C_{new}^{brush} &= \frac{C_{old}^{brush} * dstFactor + C_{sum}^{canvas} * brushperm}{(dstFactor + srcFactor)} & (5.16)
\end{aligned}$$

where the variables have the following meanings:

- ϕ_{tot}^{brush}: the total volume of the brush.
- ϕ_{tot}^{canvas}: the number of paint-filled cells occupied by the brush.
- $C(i,j,k)$: The color of the canvas in cell (i,j,k).
- C_{new}^{brush}: the new color for the brush
- C_{old}^{brush}: the previous color of the brush
- $brushperm$: the artist-controlled parameter between 0 and 1 that determines the permeability of the brush, i.e. its susceptibility to influence from the canvas color.

5.4.3 Brush Controls

There are a few parameters that control the bi-directional paint transfer operation, which are under the artist's control. Two have already been mentioned, but I summarize them below, and provide a short description of each:

- **Stain power**: (*stainpow*) Controls how much the brush color stains the canvas paint.
- **Brush permeability**: (*brushperm*) Determines how much the canvas color influences the brush color. If zero, the brush color is permanent, as in most painting programs.
- **Maximum deposition**: Determines the maximum thickness of paint that can be deposited by a single stroke.
- **Eraser mode**: This is a binary parameter that when "on" causes the brush to erase paint rather than deposit it. The mechanism behind this effect is straightforward: the brush simply reduces ϕ in voxels it occupies rather than increasing it.

Other operations are possible, but these are a few that have proven to be useful to the artists who tested the Stokes Paint system.

5.4.4 Paint filtering

As mentioned in Section 5.2, Stokes Paint approximates the non-Newtonian shear-thinning behavior of paint using three regimes. The third regime is a lower viscosity mode in which the paint flows fairly easily after being subjected to shear stress from the brush. This is achieved using filtering operations directly on the ϕ values of the canvas in the immediate vicinity of the brush.

The paint filtering operations are also useful for the purpose of maintaining a well defined and continuous surface. There are two complementary reasons to filter. First, numerical diffusion of the interface introduced by advection leads to an ill-defined surface. Second, sharp discontinuities in the phase values lead to inaccurate normals. Essentially one desires the phase field to always approximate a smoothed step function.

To achieve this, I have developed a curvature-driven smoothing filter to reduce sharp features, and a gradient-driven steepening filter to force flat regions towards either 0 or 1.

By curvature-driven, I mean the filter is applied only in regions of high curvature. In order to achieve this, one must be able to calculate the curvature of the surface. Mean surface curvature can be computed directly from the phase values as the divergence of the normals, $\kappa = \nabla \cdot \mathbf{n}$ (Osher & Fedkiw, 2002), which can be written:

$$\begin{aligned}\kappa &= (\phi_x^2\phi_{yy} - 2\phi_x\phi_y\phi_{xy} + \phi_y^2\phi_{xx} \\ &\quad +\phi_x^2\phi_{zz} - 2\phi_x\phi_z\phi_{xz} + \phi_z^2\phi_{xx} \\ &\quad +\phi_y^2\phi_{zz} - 2\phi_y\phi_z\phi_{yz} + \phi_z^2\phi_{yy})/|\nabla\phi|^3,\end{aligned}$$

where subscripts denote partial derivatives. The standard discretization of this equation using central differencing is second-order accurate.

For curvature-driven smoothing, I use a seven-point blurring kernel (which can be seen as causing a kind of paint diffusion). This kernel updates each phase value as a weighted convex combination of itself and its six neighbors, with a weights that depend on curvature:

$$\phi'_{(i,j,k)} = \frac{\phi_{(i,j,k)} + c_s \sum_{(l,m,n)\in neighbors} \phi_{(l,m,n)}}{1 + 6c_s},$$

where c_s is the smoothing amount. I have found a good choice to be $c_s = \text{clamp}((|\kappa| - 80)/100, 0, 1/6)$.

To repair smearing artifacts with gradient-driven steepening, I use a function of the form $\phi' := \phi + (\phi - 0.5) * c_p$ to push ϕ values toward the extremes of 0 and 1. I have found a good choice for the push factor in conjunction with the above smoothing function to be $c_p = \max((200 - |\nabla\phi|^2)/2000, 0)$. Note that because of the choice of thresholds, this steepening operation will only operate on the smooth regions of the field, while the smoothing operation above only operates on very steep regions, so that neither undoes the work of the other.

The filtering reduces visual artifacts and serves to recreate some of the effects of surface tension and gravity that apply in the lower-viscosity regime, effects that are not modeled in the numerical method for the high-viscosity regime.

5.4.5 Surface Extraction

For rendering, one needs to compute an approximation of the isosurface and its normals. The phase-field technique represents the full 3D topology of the fluid; however, in the case of interactive painting, a height field representation is generally acceptable for displaying intermediate results to the artist, and is much less costly to extract than a full isosurface. After a painting is finished, a complete 3D isosurface extraction method such as Extended Marching Cubes can be used. For interactive display, however, I obtain a height field from the phase values in a straightforward manner by computing one height value for each column of cells in the grid. I use a simple linear search to find the uppermost fluid cell then interpolate to estimate the isosurface location to sub-cell accuracy. Searches on successive columns can be accelerated by starting the search at the height computed for the previous column.

Once the surface location is determined, the surface normals can be computed from that extracted height field surface by finite differences, but more accurate normals can be obtained by directly computing the gradient of the phase field. The normal is simply $\mathbf{n} = \nabla f / |\nabla f|$, which can be computed with second order central differences. The two normals computed at the cell centers closest to the fluid surface are interpolated to yield the normal for rendering that cell.

5.4.6 Drying

Like dAb, Stokes Paint is able to dry. Drying is accomplished by tagging cells in the simulation as no longer being part of the fluid, but rather part of the fixed boundary, and their ϕ values are set to 1.0, as is required for all boundary cells. Since changing ϕ values near the surface will also cause the exact location of the surface ($\phi = 0.5$ level-set) to change, the second step of drying is to adjust the ϕ values of the cells just above the dried cells so that the surface extraction operation will still yield the same location for the surface.

5.4.7 The Canvas

It is desirable to model the rough texture of canvas and to capture the effect it has on painted strokes. The currently means of representing such surface texture is simply via the voxel representation of the boundary. To model a rough canvas, voxels can be set as “air” or “wall” appropriately in order to approximate any given heightfield. One can also enhance the appearance of texture by setting the canvas’ surface color using an image with lighting already pre-factored into the color.

5.4.8 The Palette

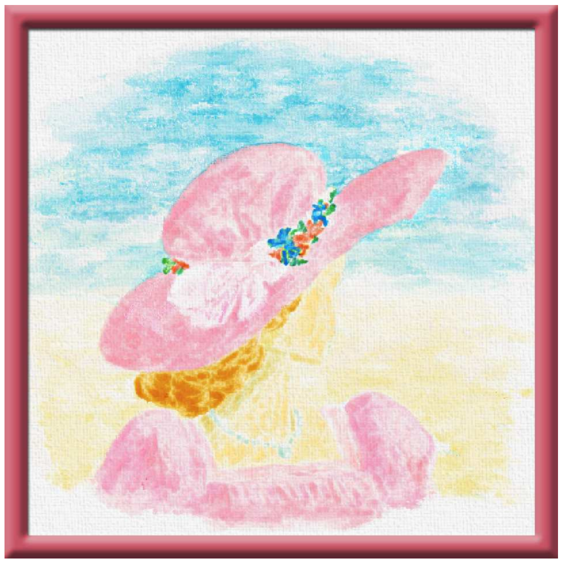
As in dAb, paint on the palette in the Stokes Paint system behaves differently from paint on the canvas. In fact, to reduce the overall memory requirements of the Stokes Paint system, the palette is implemented as a 2D paint surface. The paint blending operation used on the palette is quite similar to that used in dAb, with the exception of the brush only possessing a single color. Just as in dAb, the palette uses a special type mask texture to delimit different areas of operation: modifiable normal palette or indelible paint well. Finally, also as in dAb, the palette never runs out of paint.

5.5 Results

I have tested my implementation of this viscous Stokes-based fluid paint model on a 2.5GHz Pentium IV machine. When used for two-dimensional flow, the Stokes portion of the viscous free surface paint simulation runs at 64×64 resolution at over 70 frames per second with rendering of tracer particles. In three dimensions, the above system can compute the flow on a $32 \times 32 \times 16$ grid at 20 frames per second. The Stokes calculations are by far the most expensive aspect of the painting computations, and in particular the pressure and implicit viscosity conjugate gradient solvers. The cost of surface filtering is relatively small in comparison. Since the method is stable, the time step does not need to be reduced even when the fluid undergoes rapid motion. In contrast, a simulation restricted by the CFL or viscosity timestep conditions would not be able to keep the simulation synchronized with wall clock time, since it would have to take many smaller sub-steps when fluid velocity is large.

In order to make interactivity feasible, the fluid simulation is windowed to calculate flow only in the immediate vicinity of the brush. The active window follows the brush around as it moves. This optimization is reasonable since a very viscous fluid essentially only moves in regions in which it is agitated. I render the results by extracting a height field and normal map from the fluid as described in Section 5.3.5. For computing the color of the paint and displaying it to the artist, Stokes Paint uses the Kubelka-Munk (Kubelka, 1954) rendering method described in Chapter 7.

Several paintings created by artists with Stokes Paint are shown in Figures 5.3 and 5.4. The latter also shows a zoomed in example of some thick, impasto marks produced by the fluid model.


Eriko Baxter

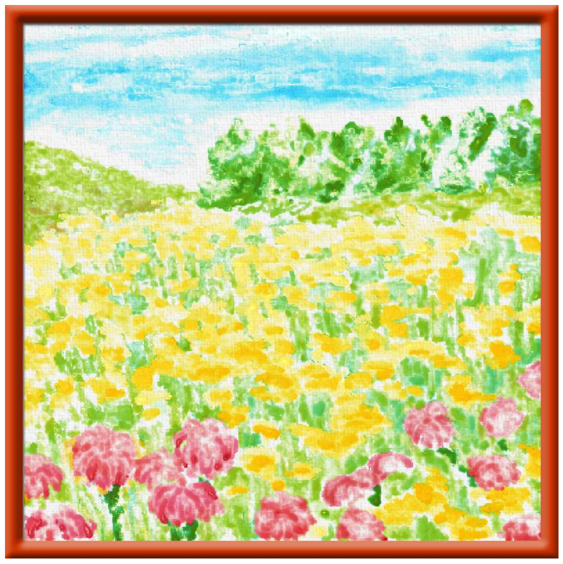
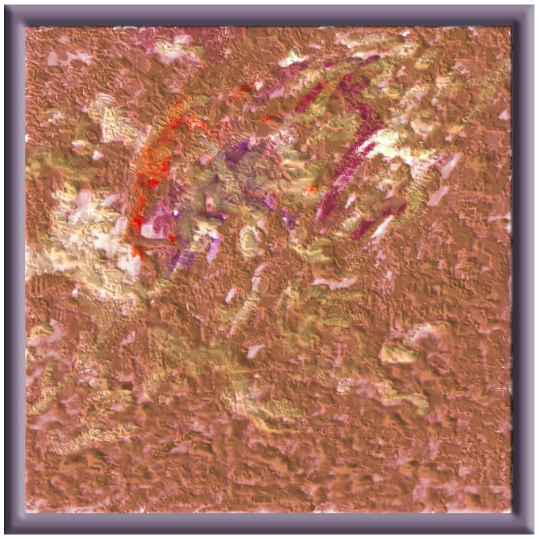

Eriko Baxter

Figure 5.3: *Paintings created by artist Eriko Baxter using Stokes Paint.*

5.6 Limitations

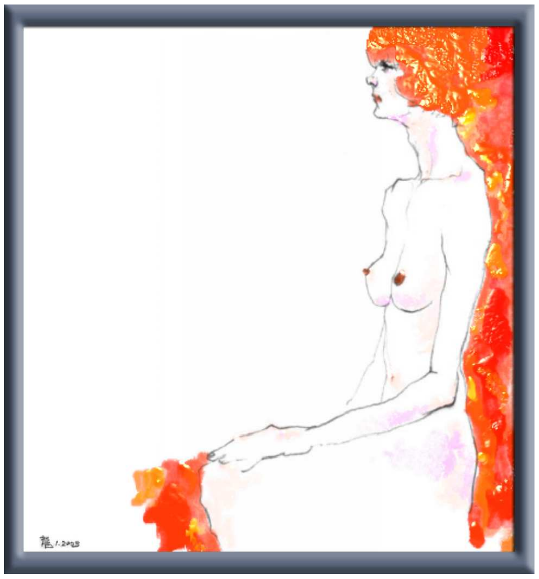
Stokes Paint has proven successful in providing painters with an interactive thick, impasto-like digital painting medium; however, there are several limitations and areas for improvement. First, the paint is currently not conserved, in large part because of the semi-Lagrangian advection of the paint. A conservative advection scheme could help here. Overall, however, the most significant issue currently is the model's relatively coarse resolution, which makes it unsuitable for very thin layers of material, or capturing very fine details. The current Stokes Paint system captures the gross physical behavior of paint well, just not the details. However at finest detail level I suspect the physical behavior is not as important, and not as noticeable to the human visual system, so I believe there is potential to augment the bulk physical model provided by Stokes Paint with lighter weight "detail behaviors" based on 2D methods, to create a paint model that combines the best attributes of a 3D fluid painting system with a 2D system. Another way of looking at the problem of resolution, though, is not as a resolution



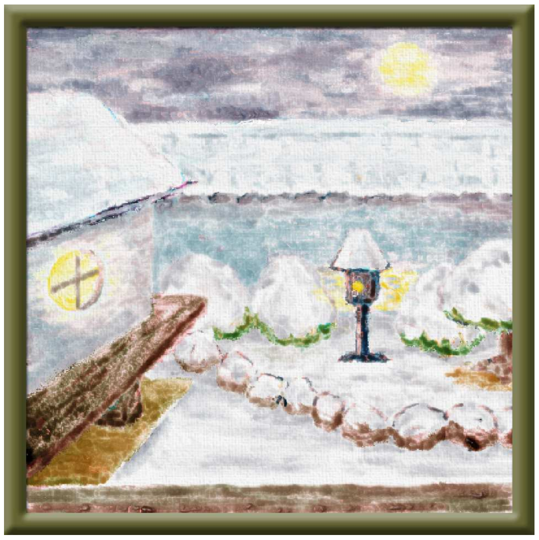
John Holloway

Haolong Ma

Andrea Mantler



Haolong Ma



Eriko Baxter

Figure 5.4: *One close-up image of some sample thick strokes in blue and red, and some more paintings created by various artists using Stokes Paint.*

problem, but a computational resource problem. The current maximum resolutions are dictated by the need for interactive speed. As the speed of computer systems improve in the future, so too should the maximum feasible resolutions, making Stokes Paint a more attractive approach as time progresses. In the mean time, one might seek a method for simulating paint that strikes a balance between the simplicity of dAb and the heavy physically-based approach of Stokes Paint. Such a paint model is precisely the subject of the next chapter.

5.7 Summary

In this chapter I have presented the Stokes Paint model. It is the first truly volumetric 3D model for viscous paint to be demonstrated in an interactive painting system. It combines a 3D Eulerian solver for the viscous Stokes equations with surface filtering techniques in order to give the painter a physically-based approximation of the behavior of real non-Newtonian paint. Though Stokes Paint is significantly more computationally intensive than dAb, with windowing it is able to operate at acceptable speeds on current systems, and more importantly, it is able to offer the artist a virtual digital medium that can be pushed and spread and otherwise manipulated much more like real paint than is possible with dAb or other existing painting systems. Though the prototype system is still just a proof-of-concept, the gallery of sample paintings already created by users, and the reactions of those users themselves, clearly attest to the artistic potential for such fluid-based painting systems.

Chapter 6

IMPaSTo: A Realistic, Efficient Paint Model

IMPaSTo is an interactive model for viscous paint media based on simplified physics, tailored for use in real-time painting systems. The three-dimensional paint surface is represented in "2.5D" using multiple height fields, with pigment concentrations and a volume (or height) stored at every pixel. The simulation uses a conservative advection algorithm that preserves both overall paint volume and pigment mass even when the paint is spread thinly. The surface painted upon is also represented with a height field, which the paint algorithm incorporates into its calculations to achieve realistic stroke texture over rough canvas or paper (see Figure 6.1). IMPaSTo allows for one wet layer of paint at a time and an unlimited number of dry layers, which can be accumulated to create optical layering effects.

The IMPaSTo paint model in many ways represents a combination of the best features of both dAb and Stokes Paint. Like dAb, IMPaSTo uses an efficient, layered 2D representation of the paint, and like Stokes Paint, the paint motion is based directly on physical principles. Compared with Stokes Paint, IMPaSTo offers significantly better resolution on current hardware both in X and Y, as well as in paint thickness. When compared with dAb, it offers significantly more realistic viscous paint behavior (see Figure 6.6). IMPaSTo also uses the Kubelka-Munk rendering technique that will be

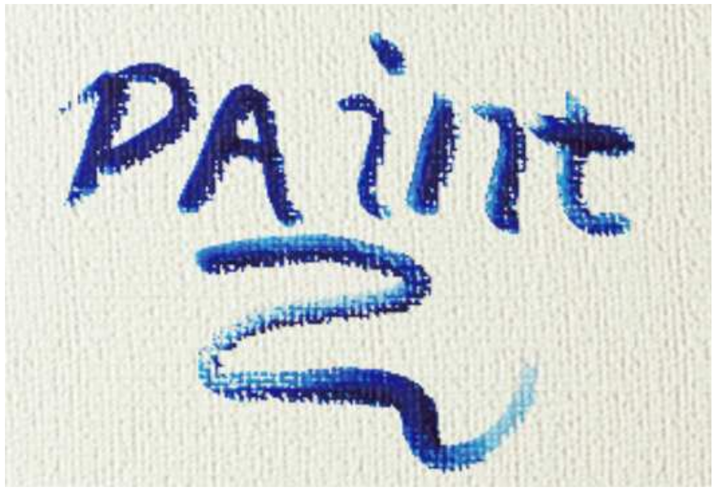


Figure 6.1: *Marks demonstrating the 3D canvas texture. The canvas surface is represented as a height field, and the IMPaSTo paint deposition algorithm takes this height into account.*

described in the next chapter, to achieve realistic mixing and compositing of paint colors.

The artists who have tried IMPaSTo have been enthusiastic about it. John Holloway, a painter by training, has written down his impressions after working with IMPaSTo over the span of a year. His comments are included in full as Appendix B.

6.1 IMPaSTo Overview

The list of painting features supported by IMPaSTo is similar to that of dAb. All of these effects are demonstrated in Figure 6.2.

One significant difference between IMPaSTo and its predecessors (both dAb and Stokes Paint, as well as other previous painting systems) is that IMPaSTo uses the graphics processor (GPU) to perform *all* of the paint simulation, as well as the rendering of the paint, rather than the CPU. By leveraging the parallel nature of the GPU, many individual brush-canvas transfer and paint rendering calculations can be performed

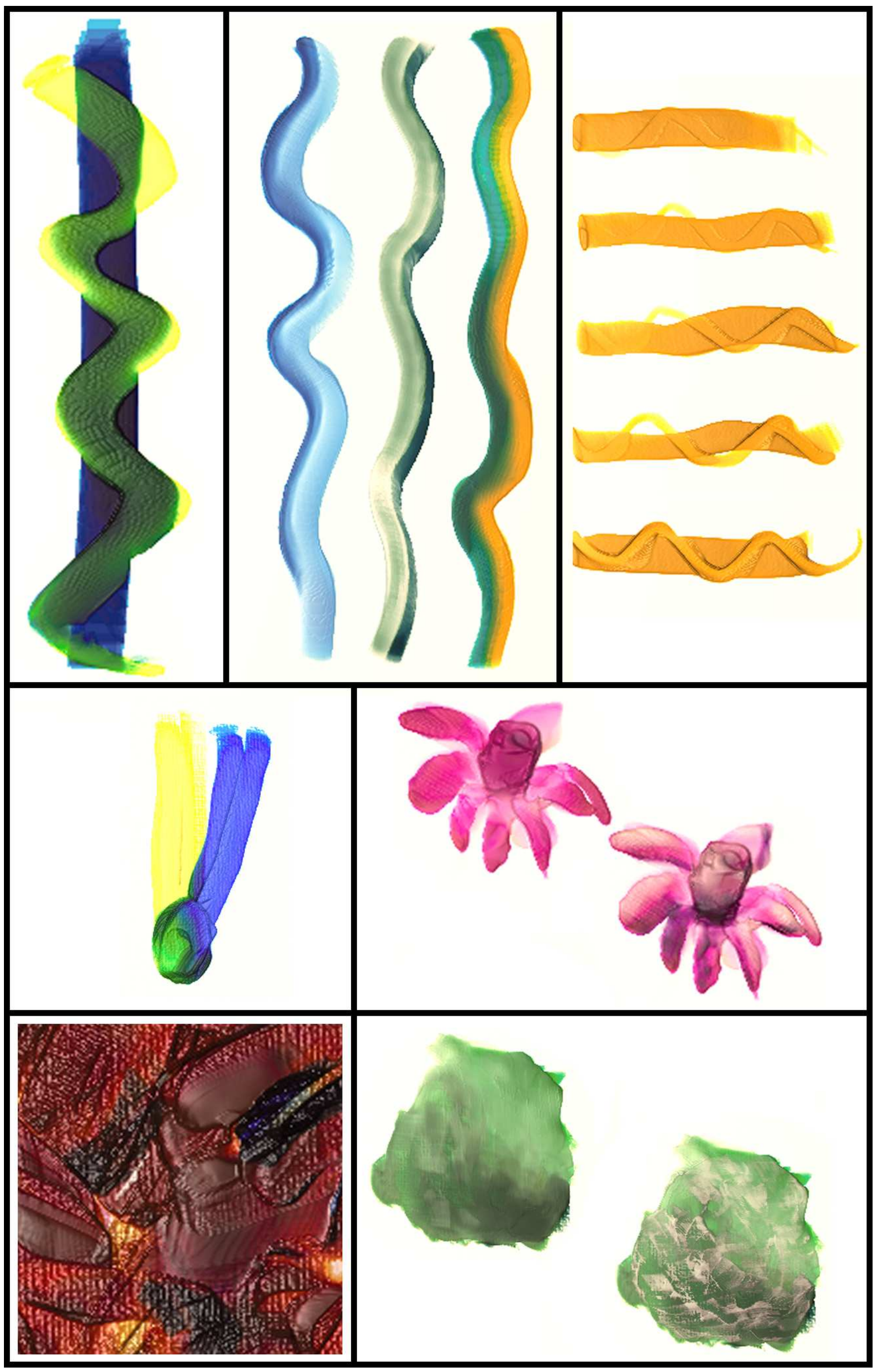


Figure 6.2: *Examples of painting features supported by IMPaSTo. From top to bottom, left to right.* Bi-directional transfer*: paint on the canvas changes the color of the brush as the curvy stroke proceeds from top to bottom;* Complex brush loading*: these strokes were made by brushes loaded with a mixture of paints;* Controlled drying*: from top to bottom the stripes have wetness 32%, 42%, 56%, 75%, and 100%. The thinner curvy strokes were made by dragging a dry brush through the wider strokes;* Blending*: Cadmium yellow and cobalt blue are blended to create green;* Glazing*: The painted flower on the left was dried and then thin glazing layers applied to change the shading;* Impasto*: Paint strokes are laid down thickly on the canvas; and* Scumbling*: The paint applied on the left was dried and then white paint was brushed lightly over the surface to catch just the peaks of the previous layer.*

simultaneously, leading to responsive, interactive performance on current systems, and also freeing the CPU for use in brush simulation and haptic feedback calculations.

The IMPaSTo interface gives the user many of the digital advantages one would expect, such as the ability to undo and redo changes at the touch of a button, and to save and manage multiple revisions. In addition, the user can instantly dry paint, or keep paint wet as long as desired, and can change the paint opacity at any time even after finishing the painting.

6.2 Paint Dynamics Principles

With the IMPaSTo dynamics algorithm, my goal was to develop the model from the start with specific paint behaviors in mind, and then to design an efficient algorithm around those principles. The general physical properties of paint interacting with a brush that I identified as critical to painting are listed in Table 6.1. While I did not conduct a formal, broad survey of painters in order to arrive at this short list, it is the result of several years of collaborations with painters and a distillation from several years of my own experience working with dAb and Stokes Paint. It should not be considered to be an exhaustive list of the properties desired of paint, but rather a proposed list of minimal requirements necessary for a paint model to have a substantial physical feel.

Figure 6.3 is a diagram of the steps in the algorithm I have developed to achieve the paint behaviors in Table 6.1. The core components of the algorithm are a conservative advection stage and carefully designed paint transfer rules. I describe each stage in detail below. The IMPaSTo paint model also allows for an unlimited number of layers of dry paint to be accumulated. I will describe the algorithm for the drying process later in the chapter.

1. Paint moves in the direction pushed
2. Paint is conserved (neither created nor destroyed)
3. Brush-canvas paint transfer requires physical contact and is greater when the brush is moving.
4. The more paint is loaded on a brush, the more will be deposited on the canvas
5. The more paint is on the canvas the more will be picked up by the brush.

Table 6.1: *The five general physical principles that govern the IMPaSTo physical paint model.*

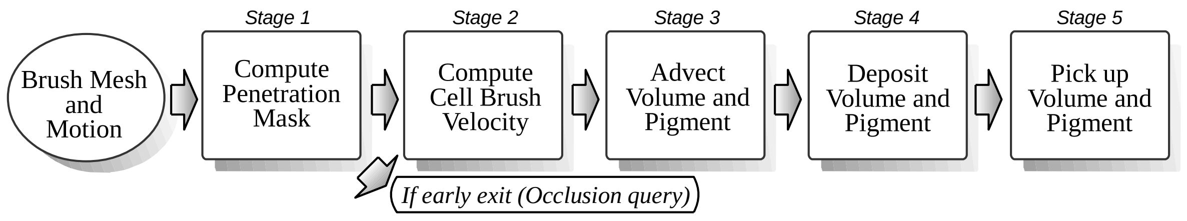

Brush Mesh and Motion
Stage 1
Compute Penetration Mask
Stage 2
Compute Cell Brush Velocity
Stage 3
Advect Volume and Pigment
Stage 4
Deposit Volume and Pigment
Stage 5
Pick up Volume and Pigment
If early exit (Occlusion query)

Figure 6.3: *Steps in 3D paint simulation. Each of the above boxes is implemented using one or two fragment programs on the GPU. After computing the penetrations I use the occlusion query to determine if any canvas fragments were in contact with the brush, and exit early if not.*

6.3 Paint Representation

IMPaSTo was designed around a 2.5D representation. Part of the motivation for returning to a layered representation, like that of dAb, was lessons learned from Stokes Paint, in which the cost of supporting a 3D grid proved to be a limiting factor on the canvas resolution with current hardware. Though a 2.5D representation cannot support true arbitrary 3D features such as bubbles or fold-overs, it can offer a reasonable approximation. Figure 6.4 shows a cross section of an actual, physical painting done in the impasto style by one of the masters of the technique, Claude Monet. Though there are some small features that do not pass the single-valued function test, from this it can be seen that a stack of height fields can still provide quite a reasonable approximation.

There are several inherent advantages to a layered representation, as well. In a 3D grid, as was used in Stokes Paint, the minimum thickness of a layer is fixed. However, the light energy transmitted through a pigmented or dyed medium tends to have an exponential relationship with thickness. The well known fog equation is an example of this type of exponential relationship. As will be seen in the next chapter, the Kubelka-Munk equations also contain terms that are exponentially related to thickness. For that reason, it would be ideal if the thickness representation were logarithmic, so that equal steps in thickness would correspond roughly to equal changes in color transmission. By using a floating point representation of thickness, IMPaSTo approaches this ideal, since floating point numbers are, in fact, a logarithmically spaced encoding of the real numbers.

As for storage of the paint, all of the relevant data to represent paint—including thickness, pigment concentrations, etc.—is stored in 2D textures. Table 6.2 lists all of the textures used, with the exception of a few temporaries for holding intermediate results. In the implementation of IMPaSTo, one canvas or brush cell is represented by a

Texture	**Use**	**Channels**	**Dim**
base canvas reflectance	R	8	canvas
painting reflectance (temporary)	R	8	tile
painting RGB composite	R	3 (4)	canvas
painting pigment concentrations	I R	8	canvas
painting thickness/paint volume (base/dry/wet)	I R	3 (4)	canvas
brush penetration on canvas (base/dry/wet)	I	3 (4)	footprint
brush velocity (x,y,z)	I	3 (4)	footprint
brush pigment concentrations	I R	8	brush
brush RGB composite	R	3 (4)	brush
brush paint volume	I R	1	brush
paint undo buffer	I	4	*

Table 6.2: *Textures used in the GPU implementation. 'I' indicates use in paint interaction and dynamics (this chapter); 'R' indicates the texture is used in rendering (Chapter 7). The "Channels" column gives the number of channels used by IMPaSTo for each texture. If more than that number are allocated due to hardware restrictions, the actual number allocated is shown in parenthesis. The "Dim" column indicates the dimensions of the texture, whether it is canvas-sized or brush-sized, etc. The size of the undo buffer is not given, since it can be as large as desired—the size determines the number of actions that can be undone.*

single texel of a texture. I have used half-precision floating point for all of the textures, except some of the small intermediate textures required in the rendering computations.

6.4 Paint Motion

In this section I present IMPaSTo's approximation of the physics of paint motion. Paint motion is driven and dominated by boundary conditions. On the one side is the paint's boundary with the moving brush, and on the other, the boundary with the stationary canvas. I simplify the actual physics of the situation by taking these boundary velocities to be the dominant terms, and deriving paint velocity in a straightforward manner from them. IMPaSTo concentrates its numerical efforts on accurately moving, or advecting, paint according to the determined velocity field. The conservative advection scheme in IMPaSTo addresses one of the limitations of Stokes Paint that was noted in the previous

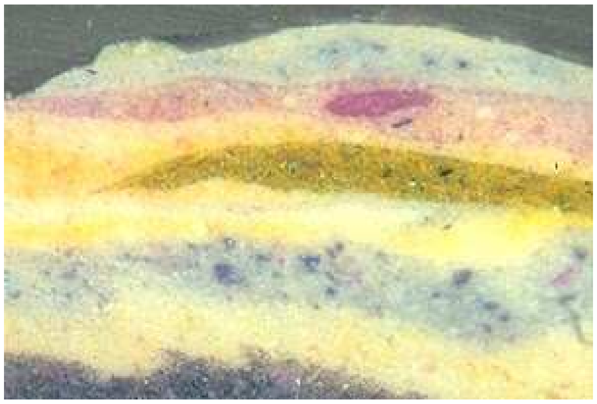


Figure 6.4: *A microscopic cross-section taken from Monet's* Water Lilies. *Note the clearly stratified structure of layers. IMPaSTo represents this type of structure using a "2.5D" stack of height fields. (©2004 The Museum of Modern Art, http://www.moma.org)*

chapter. First I will detail the advection scheme and then describe how I compute the velocity field.

6.4.1 Advection

The first two of the desired paint features listed in Table 6.1 are handled by our conservative advection algorithm, which is Stage 3 in the overall paint pipeline (Figure 6.3). Much research exists on conservative numerical solutions to hyperbolic partial differential equations. The standard text on the topic is (LeVeque, 1992). I present a basic variation of one such method here for solving the advection PDE.

The advection problem is as follows. One is given a scalar quantity q, such as the concentration of a pigment, and one wishes to determine how that quantity evolves over time under a specified velocity field $\mathbf{v}$. Mathematically, the problem can be expressed as finding the solution to the partial differential equation:

$$\frac{\partial q}{\partial t} = -(\mathbf{v} \cdot \nabla) q. \tag{6.1}$$

In one dimension the problem reduces to just

$$\frac{\partial q}{\partial t} = -v\frac{\partial q}{\partial x}, \tag{6.2}$$

and if v is constant, the solution is just $q(x,t) = q(x - vt, 0)$, i.e. the initial quantities at time 0 are just translated by vt. Upon extension into higher dimensions, the solution is not so simple, even for time-invariant velocity fields, and so numerical methods are required.

In a conservative numerical scheme for a hyperbolic conservation law, one constructs a flux function F that represents how much of the conserved quantity leaves and enters each cell of the computational grid. By ensuring the flux lost by one cell is always gained by another, one can guarantee the method will be conservative. A numerical solution to the 1D advection problem can be written in terms of flux as

$$q_i^{n+1} = q_i^n + \frac{\Delta t}{\Delta x}(F(q_{i-1/2}^n) - F(q_{i+1/2}^n)) \tag{6.3}$$

The q values are stored at cell centers (denoted as i), and are interpreted as cell average values. The total amount of q in the cell initially is thus $q_i\Delta x$ in 1D or $q_{i,j}\Delta x\Delta y$ in 2D. The fluxes are computed at cell edges (denoted as $i{\pm}1/2$) and represent the amount of material crossing that cell boundary, with positive fluxes denoting flow in the $+x$ direction.

I diverge from the typical staggered grid formulation above and instead use a cell-centered, or collocated, grid, where both velocity and advected quantities are defined at the center of each cell. The numerical scheme I use is then as follows, explained first in 1D for simplicity (see Figure 6.5). Given a discrete velocity field v_i defined at cell centers, translate each column of q by $v_i\Delta t$. The total amount in cell i initially is $q_i\Delta x$, and an amount $q_i|v_i\Delta t|$ leaves the cell under the velocity field, leaving $q_i(\Delta x - |v_i\Delta t|)$

behind. Note that in order for cell i to not lose more flux than it originally possessed, one must have $\Delta x - |v_i|\Delta t > 0$. This imposes a limit on the maximum velocity possible, namely $|v_i| < \Delta x/\Delta t$, commonly known as the CFL condition. Cell i may also gain flux from its neighbors. The amount gained from cell $i-1$ is either 0, if $v_{i-1} < 0$, or $q_{i-1}v_{i-1}\Delta t$ otherwise. A similar expression exists for flux gained from cell $i+1$.

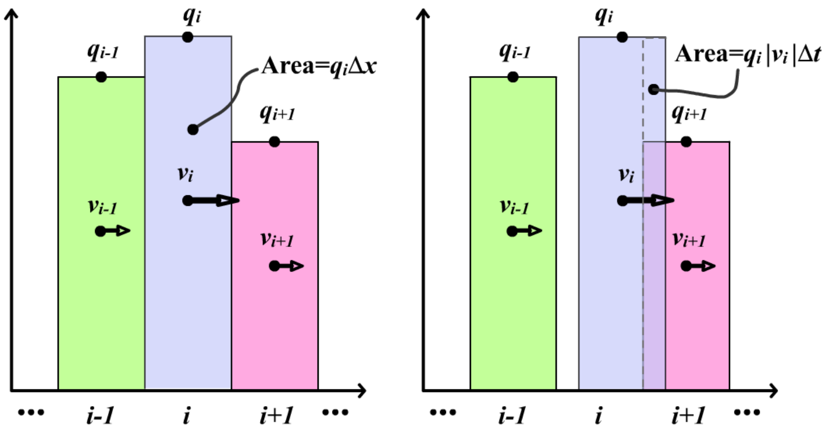


Figure 6.5: *Conservative advection flux computation in 1D. Cell $i+1$ gains a flux of $q_i|v_i|\Delta t$ from cell i, and cell i loses the same.*

The same basic scheme can be carried out in higher dimensions. For instance, in 2D, given a cell-centered velocity $\mathbf{v}_{i,j} = (u, v)$ one can treat the column of q as moving by $u\Delta t$ in the x direction and $v\Delta t$ in the y direction and determine flux donated to other cells just as was demonstrated above in the 1D case.

As mentioned, IMPaSTo represents paint as a stack of height fields. I use a 2D advection scheme, and handle the third dimension by treating the volume of paint in each cell as another scalar to be advected along with pigment concentrations. That is, the height field is advected according to the fluid velocity just like pigment concentrations.

I have found one modification to the above algorithm to be useful. With the algorithm just as described above, it is quite possible to completely advect paint off of a particular canvas cell. This gives the canvas the appearance of being heavily sized[1].

[1]Size is an animal based glue used to seal the surface of a canvas.

To model a canvas surface with more adhesion and absorption, I simply do not allow the advection step to remove all the paint in a cell. The computed flux quantity is clamped to leave at least a parameter-defined minimum quantity behind. Using this parameter, the artist can control the amount of “virtual size” to apply to the canvas (though in the underlying simulation it is more a matter of adding anti-size).

6.4.2 Computing Velocity

The preceding description of our advection calculation was predicated on the *a priori* knowledge of a velocity field to use. In real painting this velocity field comes from several sources. As mentioned, the main sources are the friction and constraint forces imposed by the brush on the one side of a layer of paint, and by the stationary canvas on the other. Any viscid fluid will have zero slip (tangential) velocity at the interface between the fluid and a solid boundary. So during a paint stroke, within the thin layer of paint trapped underneath the brush, the paint in contact with the brush has the brush’s velocity, while paint in contact with the canvas has the canvas’s velocity. Since paint is a continuum, all possible velocities between zero and the brush speed must exist within the layer of paint. Thus as a first approximation, a reasonable 2D velocity to assign the paint is 1/2 of the of the brush’s tangential velocity relative to the canvas surface. This kinematic brush velocity, v_b, is the first component of the total velocity used. This velocity only applies to cells in the canvas surface that are in contact with the brush as determined in Stage 1 of the computation pipeline (Figure 6.3).

But paint is not just two-dimensional, and although painting a stroke is primarily a motion in the 2D canvas plane, the out-of-plane motion and vertical force of the brush are also important. Paint is an incompressible fluid and it is affected by internal pressure forces. For instance, when a force is applied downward from above, the pressure field that develops internal to the paint induces a flow in the direction of the negative pressure gradient. In other words, it causes the paint to flow outward in any unconstrained direction. To model this "squishing" behavior I use a simple heuristic. First, for every cell in the 2D paint grid where the brush penetrates the heightfield surface, I compute the amount of penetration, p (done in the first stage, Figure 6.3). Next I compute the 2D gradient of the penetration amount, ∇p, and define a heuristic "pressure"-driven velocity to be a constant times that value, $v_p = -c\nabla p$. This pressure driven velocity, v_p is then simply added onto to the brush velocity v_b to get the total velocity at each cell of brush-canvas contact. Note that this can be seen as an approximation of the pressure term on the right hand side of the full Navier-Stokes equations for fluid motion:

$$\frac{\partial \mathbf{v}}{\partial t} = -(\mathbf{v} \cdot \nabla)\mathbf{v} + \nu \nabla^2 \mathbf{v} - \nabla p + \mathbf{F}, \quad \nabla \cdot \mathbf{v} = 0 \tag{6.4}$$

where I substitute amount of penetration for the amount of internal pressure, assuming the two to be roughly proportional.

When laying down a stroke, IMPaSTo moves the brush along the stroke path no more than one cell width at a time to ensure that v_b does not violate the CFL condition. Also, after adding in v_p, the final x and y velocity components are clamped to be within [-1,1], to prevent unusually large values of v_p from causing a violation of the CFL.

6.5 Paint Transfer

The paint transfer algorithm is responsible for determining how much paint moves from the brush to the canvas and vice versa. Each of the remaining desiderata in Table 6.1 is handled via the transfer algorithm (Figure 6.3, Stages 4 and 5).

The first assumption I make is that at any given cell where brush-canvas contact is occurring, the transfer flow is uni-directional. That is to say, if paint is being deposited onto the canvas at a particular cell, it cannot also be loading into the brush simultaneously. Note that since this is a *per-cell* determination, it is still possible that at any given instant some parts of the brush are loading paint, while others are depositing paint. The direction of the flow is determined by whether there is more paint on the canvas, a_c, or more paint on the brush, a_b. Rather than the full amount of paint in the canvas cell, I define a_c to be the full volume times the fraction by which the brush is determined to be penetrating the paint surface in Stage 1 (Figure 6.3). In this way, one can still deposit paint on the surface of a thickly covered canvas by brushing lightly.

Algorithm 3 gives the full calculation used. First, no paint is transferred if the brush is not in contact. Then, the direction of flow is determined by whether a_b or a_c is greater, and the base amount of flow is computed as a fraction of either a_c or a_b depending upon the direction. The base transfer amount is then modified in several important ways. The transfer is gently cut off to zero when a_c is nearly equal to a_b to prevent developing unstable oscillations of the paint transfer back and forth. Next I cut off the transfer amount gradually if the brush velocity is below a threshold, to account for the need for some sliding friction to "pull" paint out of the brush. Without this, the brush appears to ooze paint unnaturally. Finally, the transfer amount is clamped to a maximum value to make the paint transfer more even. The pseudocode of Algorithm 3 shows all of these steps and includes parameter values I have found to work well. To put the constants in context, the paint thickness for a thin painting is typically around 0.001 units and around 0.1 for a thicker style. Velocity is in terms of cells per timestep, so 1.0 is the maximum possible velocity component, according to the CFL condition.

When paint is transferred either direction, or is moved by the advection algorithm, I compute the new pigment concentrations on the affected brush or canvas cells by a simple volume-weighted average.

```
$(x_{bc}, x_{cb}) \leftarrow$ ComputeBrushTransferAmount$(a_c, a_b, v)$
    $\triangleright$ let $a_c$ be amount of paint penetrated in canvas cell
    $\triangleright$ let $a_b$ be amount of paint on corresponding brush cell
    $\triangleright$ let $v$ be tangential velocity of brush
    $\triangleright$ let $x_{bc}$ be the amount transferred from brush to canvas
    $\triangleright$ let $x_{cb}$ be the amount transferred from canvas to brush
    $\triangleright$ let XFER_FRACTION $= 0.1$
    $\triangleright$ let MAX_XFER_QUANTITY $= 0.001$
    $\triangleright$ let EQUAL_PAINT_CUTOFF $= 1/30$
    $paintDiff \leftarrow a_b - a_c$
    $equalPaintCutoff \leftarrow$
        **clamp**$(|paintDiff|/$EQUAL_PAINT_CUTOFF$, 0, 1)$
    $velocityCutoff \leftarrow$ **smoothstep**$(0.2, 0.3, ||v||)$
    $xferDir \leftarrow$ **sign**$(paintDiff)$
    **if** $xferDir > 0$
        **then** $amt \leftarrow a_b$
        **else** $amt \leftarrow a_c$
    **end**
    $amt \leftarrow amt *$ XFER_FRACTION
    $amt \leftarrow amt * equalPaintCutoff * velocityCutoff$
    $amt \leftarrow$ **clamp**$(amt, 0,$ MAX_XFER_QUANTITY$)$
    **if** $xferDir > 0$
        **then** $(x_{bc}, x_{cb}) \leftarrow (amt, 0)$
        **else** $(x_{bc}, x_{cb}) \leftarrow (0, amt)$
    **end**
```

ALGORITHM 3: *The paint transfer algorithm*

6.6 Paint Drying

The IMPaSTo paint model supports the drying of wet paint in order to allow the user to build up paintings out of many layers of paint. The process IMPaSTo supports is similar to that of dAb and Stokes Paint, which is different from the drying of actual paint. In IMPaSTo wet paint is always completely wet, and dry paint is completely dry. However, these are the two extremes that are typically desired by the artist. Given a layer of wet paint IMPaSTo allows fractional drying of that layer as a percentage of the overall thickness. So if the user elects to dry the paint by 25 percent, IMPaSTo creates a new dry layer out of the bottom quarter of the wet layer, leaving 3/4 of the wet paint in tact.

While the memory and processing requirements to support an unlimited number of wet layers of paint would be prohibitive, it is possible to support as many dry layers as virtual memory will hold, since they are static, and for the purposes of painting a stroke, can be treated the same as the static base canvas texture. Thus IMPaSTo only needs to know their combined thickness, which can be computed just once and stored in a texture. IMPaSTo must also maintain the combined reflectance of all the dry layers for use by the rendering algorithm, which will be discussed in detail in Chapter 7.

When a new layer is dried, the combined thickness and reflectance information is just a function of the current composite thickness and reflectance, and the thickness and pigment concentrations of the new dry layer. The computation involved in updating the reflectance will be discussed in more detail in the next chapter, and is the same process that will be shown in Figure 7.8. Note that even though the runtime system does not need the dry layer's pigment information once a drying operation is complete (just its reflectance), IMPaSTo must store the pigment data nonetheless for use when changing the lighting spectrum.

6.7 The Palette

The IMPaSTo palette is very similar to that of dAb. A special type-code texture differentiates the various palette regions: normal palette, paint well, and brush cleaner. Like the other palette implementations, the IMPaSTo palette never runs out of paint. Currently IMPaSTo conserves the somewhat limited graphics memory by using a palette with lower resolution than the main canvas. Other than that, and the qualities just mentioned, the palette acts just like an IMPaSTo canvas. The fragment programs used to implement the palette paint behavior are the exact same programs used by the canvas, with just a few minor modifications implemented via conditional compilation preprocessor directives.

6.8 Brush Modes

I have implemented several different brush modes that give the painter a greater range of stylistic choices when painting. Most of these modes are just simplifications of the full IMPaSTo paint dynamics pipeline (Figure 6.3). For instance, by disabling writes to the brush volume texture, it is simple to create a brush that never runs out of paint.

The following is a list of the brush modes implemented in IMPaSTo and a short description of how each affects the brush transfer and interaction with the paint:

- **Normal**: The standard, full painting pipeline as shown in Figure 6.3.
- **No push**: The advection stage is disabled, so that paint can be transfered to the canvas or picked up by the brush, but the brush cannot push the paint.
- **Sponge**: Update to the brush volume texture is disabled. This causes the brush to act similar to a high capacity sponge in that, when it is full, it can deposit paint indefinitely, but when it is empty, it can absorb paint indefinitely.
- **Sponge, no push**: A combination of the "no push" and "sponge" modes.
- **Paint Marker**: In this mode, the amount of paint transferred from the brush to the canvas is determined by a simple "marker flow" parameter that is under artist control.
- **Paint Sampler**: This mode emulates the "eye dropper" tool common to many painting programs. In paint sampler mode, the canvas is not modified at all, but

the brush color is changed to whatever canvas color it comes in contact with. Each texel of the brush performs the color sampling, so the sampler mode can be used to load a complex blend of paint onto the brush.

- **Fill**: This coats the entire canvas with a thin layer of paint using the average color currently loaded on the brush.

- **Eraser**: In fact there is no eraser mode, but there is a single key that clears the brush of all paint. If this key is held down as the brush is moved over the canvas, the effect is that paint is "sucked into" the brush, much like a paint vacuum cleaner.

Though none of these supplementary modes is really necessary to create paintings, they can make achieving certain effects simpler, if the artist is interested in taking the time to learn them.

6.9 GPU Implementation

I have implemented the IMPaSTo algorithm using NVIDIA GPUs and the Cg language for vertex and fragment programming. All of the relevant data is stored in textures as described in Section 6.3, and these textures are operated on by fragment programs. In my implementation, one canvas or brush cell is represented by a single texel of a texture. In the GPU implementation, each of the stages in Fig 6.3 is performed by one or two fragment programs.

The same issues with generating footprints and attribute mappings, as discussed in Chapter 4, are relevant to IMPaSTo as well. IMPaSTo generates the mappings in the same way as dAb; however, the operations are all implemented in fragment programs, so that no readback from graphics memory to system memory is ever necessary.

Computing the brush to canvas mapping really is a rasterization problem, making this part of the problem an ideal fit for the GPU. To implement this on the CPU one would essentially have to write a software rasterizer. The rest of the algorithm also maps well to the GPU since all of the operations (e.g. penetration gradient and advected flux) can be formulated locally so that they depend at most on their immediate neighbor fragments.

Another detail worth mentioning is the tiling implementation. Most image manipulation programs use some form of tiling to speed up operations. For undo as well as for rendering computations, I break the painting up into tiles of size 64×64. When a brush stroke is about to modify the data in a tile, the original data must be backed up somewhere to provide the ability to undo the action. In dAb and Stokes Paint, which ran mostly on the CPU, this undo information was stored in system memory, but for a GPU-based implementation, copying to system memory introduces an unacceptable penalty due to the slow read-back from the GPU memory. Instead I allocate one additional "undo texture" in the graphics card memory and dynamically allocate undo tiles out of it. I am thereby able to use fast texture-to-texture copies on the GPU to save the necessary undo data for each tile. The use of tiling also speeds up the rendering computation tremendously, because only dirty (that is, modified) tiles need to be updated.

One other important reason to use tiling is for cache coherence. If the entire image buffer is laid out linearly in row-major order, then the memory access pattern in updating a brush footprint on the canvas will have poor cache coherence. The brush may only touch 30×30 pixels, which could fit in the cache fine, except each of the 30 rows in the footprint are spread widely in memory, and the cache will likely pull in much unrelated data from the same rows in trying to access the entire footprint. The problem just gets worse the wider the canvas. If instead the canvas is stored as tiles, then only the tiles through which the brush passes need to be in the cache, and normal locality-of-reference schemes for caching data will perform well. Fortunately, most GPUs automatically store textures in a tiled manner for efficient localized access, so the author of a GPU painting program need not worry. This is one aspect of coding a painting system that is actually simplified by using the GPU.

6.10 Results

I have tested the IMPaSTo viscous paint model implementation on a 2.5GHz Pentium IV machine with an NVIDIA GeForceFX 5900 Ultra graphics card. Note that the CPU speed is not critical for interactive response, and IMPaSTo has also been used successfully on CPUs of less than 1GHz. The time required to draw a stroke is almost completely dominated by the physical paint model, since the cost of the optical model is greatly reduced by the tiling and lazy evaluation. For a brush footprint of approximately 26×26, the paint simulation pipeline shown in Fig. 6.3 is able to run about 116 times per second, processing an average of 77,000 canvas cells per second (i.e. texels/sec). For a larger brush footprint of about 88×88, the IMPaSTo pipeline can only run 68 times per second, but texel throughput increases to 519,810 canvas cells per second. At these speeds, IMPaSTo is able to keep up with the user for strokes of moderate speed. For faster strokes the input data is buffered and the stroke lags slightly behind the user. The improved texel throughput for bigger brushes is a strong indication that much of the time is spent in per-pass setup overhead and GPU context switches.

I have integrated the IMPaSTo model with the painting system framework created initially for dAb. It provides the user with a large canvas, and, similar to Stokes Paint, runs the fragment programs only in the bounding rectangle of the region of brush contact. As in dAb and Stokes Paint, tiling is used to speed up rendering. I mark

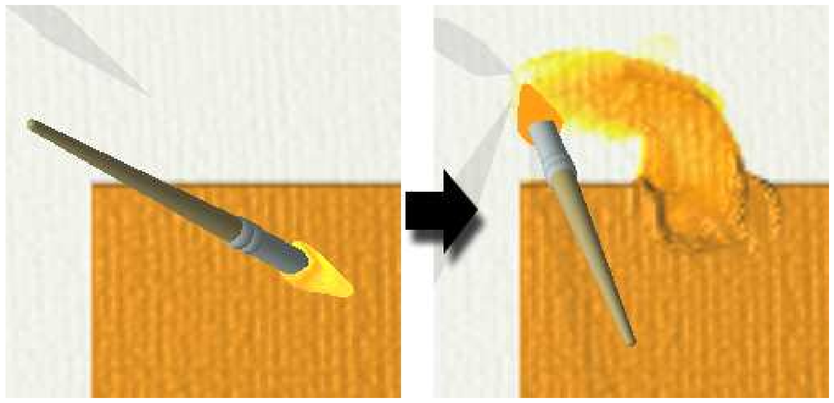


Figure 6.6: *An example of the paint motion generated by the IMPaSTo model when the brush interacts with the paint on the canvas.*

the canvas tiles through which a stroke passes as dirty and recompute reflectances and relight the canvas on a tile-by-tile basis as needed when time permits.

The many figures in this chapter show examples of various styles, effects and paint textures that the IMPaSTo paint model is capable of creating. These paintings demonstrate the range of paint-like effects the model is capable of achieving. Most of these paintings were created by amateur artists within a couple of hours, without much training or elaborate instruction.

A simple example of the type of paint motion achieved by the IMPaSTo paint model is shown in Figure 6.6.

6.11 Limitations

There are currently several limitations to the paint simulation approach presented in this chapter. First, the maximum resolution IMPaSTo is able to achieve on current hardware without sacrificing interactive performance, though improved compared with Stokes Paint, is still limited due to computational costs. However, I believe that many speed-ups to the GPU implementation are still possible at many levels, and at the same time GPU performance continues to increase as well. Beyond the computational cost;

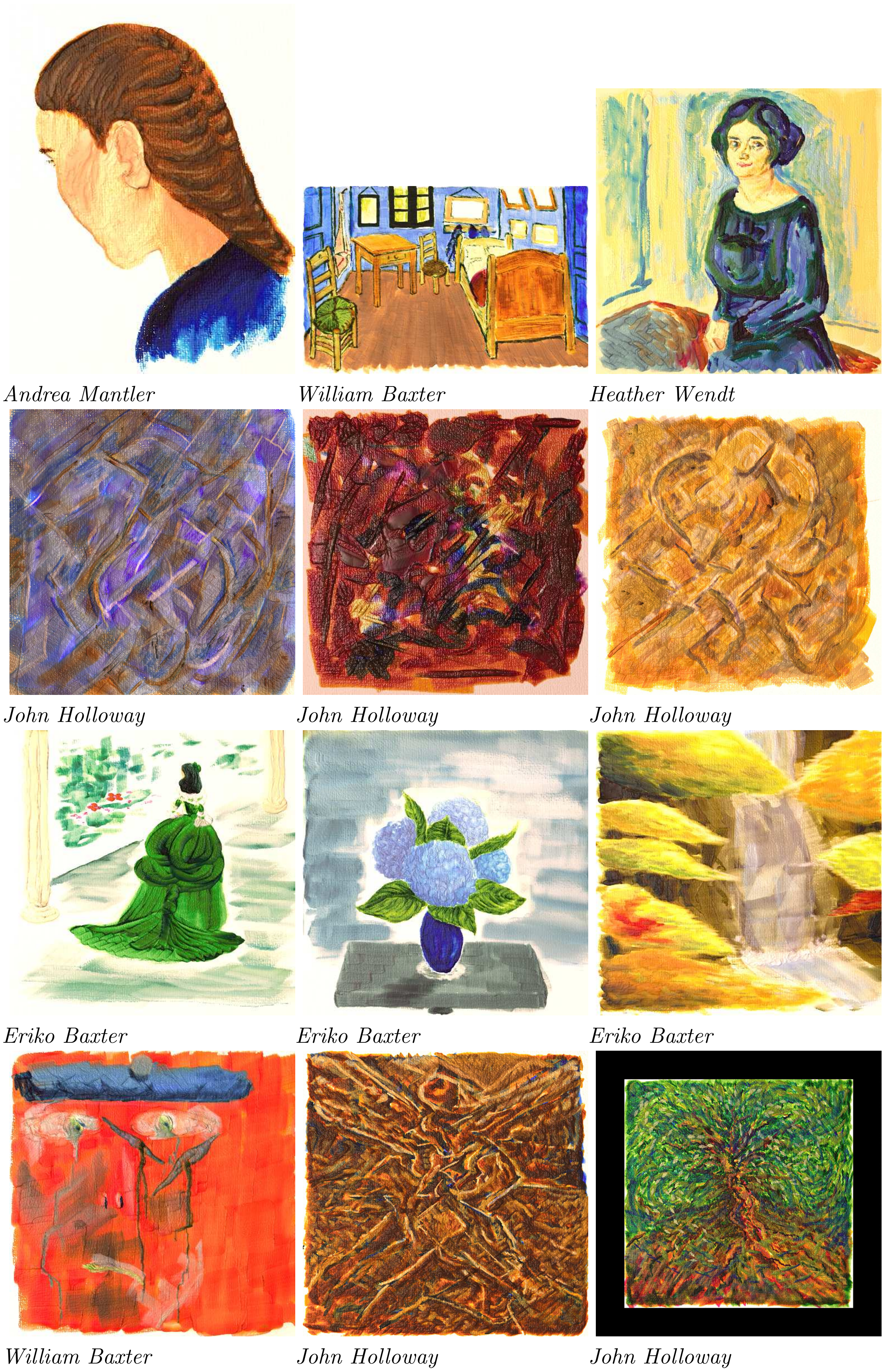


Figure 6.7: *Paintings created by various artists using IMPaSTo.*

however, there is the issue of GPU memory capacity and support for large textures. The maximum GPU memory currently available is 256MB, which amounts to just 64 channels of 32-bit floating point data at a resolution of 1024x1024. The maximum texture size supported by current GPU drivers is 4096×4096, which could have only 4 32-bit channels. The current set of textures used by IMPaSTo requires 20 channels of data for the canvas (though 16-bit precision does suffice for most of that). It can be seen then that memory size restrictions are a consideration. In order to have a larger canvas, some sort of paging scheme would be required, which is not currently implemented.

One example a possible speed-up is the use of multiple render targets (MRT) on more recent GPUs. These GPUs allow more than one texture to be updated in a single pass. I could reduce the number of redundant computations in the current implementation by using this feature. The advection step in particular could benefit, as it currently recomputes sampling weights from scratch for each texture that must be advected. Instead, with MRT only a single advection pass would be necessary. I also have yet to put any significant effort into aggressively limiting the precision of calculations wherever possible. Most values are using 16-bit floats currently, but some could function with just 8-bits of fixed precision. Such reductions in data size can offer benefits both in terms of reduced computation as well as reduced bandwidth usage.

The main limitation of the dynamics model itself is that it does not incorporate any notion of gravity. This means that it is possible for paint to pile up in physically impossible ways, such as a thin vertical column of fluid. In real paint this behavior would certainly exceed the yield limit of the paint, causing it to flow. One way to capture this effect would be to include a diffusion model, as was used by (Cockshott et al., 1992), to cause paint to flow from cell to cell when a height gradient threshold is exceeded between those cells. In fact most of Cockshott's autonomous dripping and diffusing model could be incorporated into IMPaSTo if desired.

Another limitation of the dynamics is that the simple local pressure approximation cannot guarantee that the fluid will flow in the proper direction. For instance, given a brush shape with a concavity, the pressure approximation will cause fluid to move to the center of the concavity where it will just accumulate. This is due to the fact that the pressure approximation I propose uses no global information. With a full Poisson solver for pressure, information can propagate globally rather than just to the neighboring cell. However, a Poisson-solver is a fairly expensive operation, and the approximation in IMPaSTo is fast, and works quite well for convex brushes.

6.12 Summary

In this chapter, I presented IMPaSTo, the third in a series of interactive paint models for the oil- or acrylic-like paints used most commonly in fine art painting.

I have presented

- Five principles of paint motion that represent a set of necessary conditions for any physical paint model.

- A physically inspired dynamics approximation for paint that follows the five principles.

- A paint dynamics model featuring a conservative, paint-volume preserving advection scheme, and realistic brush-canvas paint transfer rules.

- A GPU implementation of the dynamics model, which allows IMPaSTo to run at excellent speeds on recent graphics cards.

Finally, I have graphically demonstrated the wide range of effects of which the model is capable, both through specific images of particular effects as well as through the gallery of paintings created by artists with the IMPaSTo system. Through these demonstrations, I have shown that this paint model is truly capable of serving as a realistic interactive digital impasto medium for painting.

Chapter 7

Paint Rendering

I have tried to express the terrible passions of humanity by means of red and green.
— Vincent van Gogh, on his painting *The Night Café*

The previous chapters have described three approaches to modeling the motion of paint. Chapter 4 also presented a simple additive linear color model for mixing and layering paint. In this chapter, I will describe techniques to achieve much more realistic rendering of paint in real-time. The techniques are based on the Kubelka-Munk pigment mixing and layer compositing equations (Kubelka & Munk, 1931; Kubelka, 1948; Kubelka, 1954). For accurate color manipulation, I propose to sample reflectances using eight color components, rather than the standard three samples of real-time graphics, and to compute RGB colors from those samples using Gaussian quadrature. Additionally, I describe methods for measuring real paint samples with a spectroradiometer and obtaining Kubelka-Munk parameters from those measurements via quadratic optimization.

A rendering method for paint should be able to compute RGB colors for display based on several factors: the color and intensity of light sources, the canvas reflectance and texture, the pigments in the painting, their concentrations, and the geometry of each layer of paint. In general, since light is a function of wavelength, a complete and accurate description of the color must not ignore this fact. My rendering method encompasses all of these factors.

The paint rendering method I present in this chapter has the following features:

- Accurate rendering of pigmented mixtures based on the Kubelka-Munk diffuse reflectance equations(Kubelka & Munk, 1931; Kubelka, 1948; Duncan, 1940).
- Accurate rendering of translucent glazes of paint based on Kubelka's layer compositing equation (Kubelka, 1954).
- Paint coefficients measured from real-world oil paints.
- Novel eight-channel custom color space based on Gaussian quadrature.
- Full-spectrum lighting allowing a painting to be re-lit by any full-spectrum illuminant.
- Real-time realization using programmable fragment shaders on graphics hardware.

Together these features combine to deliver a more realistic interactive paint rendering system than has ever been presented previously.

The next few sections give an overview of paint, color science and other relevant previous work. Then I describe my rendering technique in detail and finally demonstrate some results achieved by my paint rendering techniques.

7.1 Paint Composition and Optical Properties

Paint is made by mixing finely ground colored *pigments* with a translucent *vehicle*. Linseed oil is the vehicle typically used in oil paints, while in acrylics it is a polymer emulsion. Both are nearly neutral in terms of coloration. The pigment particles are suspended in the medium and are the factor primarily responsible for a paint's color. Pigments are generally either inorganic compounds, such as the titanium oxide used in titanium white, or organic compounds, such as anthraquinonoid, which is used in alizarin crimson.

The sizes of pigment particles typically used in paints range from 0.1–50μm (100–50,000nm)(Callet, 1996). The size of the particles makes a significant difference in the optical properties of the paint as a whole, as does the ratio of the index of refraction of the particles to the index of the medium. With large particles, light reflects and refracts much as with any macroscopic surface. As the particle size goes below about 10 times the wavelength of the incident light (4000–7000nm), however, scattering effects begin to become significant. Many mathematically sophisticated theories of scattering exist, such as Mie theory, which can predict how the distribution of scattered light changes both due to particle size and the relative indexes of refraction. High scattering is desirable especially in white paints, because more scattering means increased opacity, or hiding power, so that fewer coats are necessary to cover the surface. According to Judd and Wyszecki, the optimal particle size for scattering is about 160–280nm, about 0.4 times the wavelength of visible light (Judd & Wyszecki, 1963, p.381).

7.2 Overview of Color

This section gives a brief introduction to color science and colorimetry. More extensive treatments can be found in texts such as (Evans, 1948; Wright, 1958; Judd & Wyszecki, 1963; Wyszecki & Stile, 1982; Hall, 1989). Kenneth Fishkin's dissertation also presents a good overview of color science topics important to computer graphics researchers (Fishkin, 1983).

Visible light is electromagnetic radiation with a wavelength in the range of about

400nm to 700nm. All color we see is the result of this radiation stimulating the special color receptor cells in our retinas, called *cones.* Humans have only three distinct types of cone, called S, M, and L, which, roughly speaking, are good at sensing blue, green, and red light, respectively. In actuality, there is a large amount of overlap in the spectral sensitivities of the three types of cones as can be seen in Figure 7.1.

Given a light stimulus, the overall degree of excitation of cones in the eye is essentially given by the integral of the light at all wavelengths across the spectrum, weighted by the sensitivity functions:

$$\int_0^\infty P(\lambda)S(\lambda)d\lambda, \tag{7.1}$$

$$\int_0^\infty P(\lambda)M(\lambda)d\lambda, \tag{7.2}$$

$$\int_0^\infty P(\lambda)L(\lambda)d\lambda, \tag{7.3}$$

where P is the spectral power distribution of the stimulus, and S, M and L are the sensitivity functions for the three types of cone. The color perceived depends on the relative excitation of each type of cone. Color vision is a very complex phenomenon, and many psychophysical factors can influence the perception of color. These factors are beyond the scope of this dissertation, so from this point on it will be assumed that the perception of color can be completely described by Equations 7.1–7.3.

A result of the trichromatic nature of our vision is that the entire range of perceivable colors can be compactly described using just three numbers rather than infinite-dimensional spectral distributions (Judd & Wyszecki, 1963). A tristimulus color space can be defined, for instance, by a triplet of numbers that represent the intensities of three colored lights. Let $\hat{r}$, $\hat{g}$, and $\hat{b}$, be the spectral energy curves associated with the three lights. The lights can be of any colors, so long as they all appear to be different colors to an observer. Each light individually excites the S, M, and L cones to differing

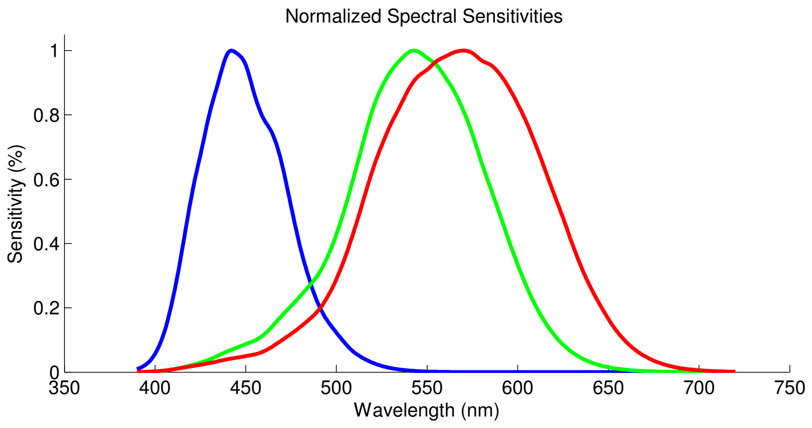


Figure 7.1: *The relative spectral sensitivities of the cones in a typical human retina. From left to right, the S, M, and L cone sensitivities. (Data courtesy: http://cvrl.ioo.ucl.ac.uk/database/data/cones/linss2_10e_1.txt)*

degrees[1]. Given these three lights, we can create new spectra by combining them with different intensities, say R, G, and B, respectively. Lights work additively, so the final spectrum created will be $R\hat{r} + G\hat{g} + B\hat{b}$, and this will be perceived as a new color. By varying the values of R, G, and B, many different colors can be produced. A natural question to ask then is, given a particular target color, what values of R, G, and B are required to generate the same color response in an observer? In general the exact spectrum of the target cannot be duplicated by changing R, G, and B; however, the color sensation can be. This is known as the color matching problem. The standard experimental setup for determining color matching functions is to have subjects control the intensities of three lights in order to match the color of a sample, and repeat this many times with many different samples and subjects, then average the results to arrive at color matching curves for a "standard observer". Given the color matching curves, say $\bar{r}(\lambda)$, $\bar{g}(\lambda)$, and $\bar{b}(\lambda)$, which describe how much of each light is required to match a

[1]Note, as an aside, that because the sensitivity functions overlap, it is not possible to create a stimulus that excites only one type of cone. In mathematical terms means that the cone sensitivities do not form an orthogonal basis for perceived color.

particular reference stimulus $P(\lambda)$, we can compute R,G, and B:

$$\begin{aligned} R &= \int_0^\infty P(\lambda)\bar{r}(\lambda)d\lambda \\ G &= \int_0^\infty P(\lambda)\bar{g}(\lambda)d\lambda \\ B &= \int_0^\infty P(\lambda)\bar{b}(\lambda)d\lambda \end{aligned}$$

Unfortunately, when using combinations of visible lights there are always some colors that cannot be reproduced exactly. To match certain visible colors, negative amounts of one or more of the lights will be required, regardless of the lights chosen. Since negative light is not physically possible, these colors cannot be matched with a three-light setup. This issue of unrealizable colors does not nullify the utility of color spaces defined by triplets of visible primaries—color television sets and computer monitors are a prime demonstration of the contrary. However, it does create inconveniences when performing color calculations.

To eliminate the inconveniences associated with negative values in tristimulus color spaces, the CIE (Commission Internationale de l'Éclairage) in 1931 defined a set of non-negative color matching functions, and associated non-negative tristimulus values[2]. The matching functions $\bar{x}$,$\bar{y}$,and $\bar{z}$, are shown in Figure 7.2. Note that all three functions are non-negative across the spectrum, ensuring that integrals of light intensities weighted by these functions will always be non-negative as well. The drawback of having a color system in which all visible colors can be represented with non-negative numbers is that the three primaries no longer correspond to physical colors. The color coordinates of a

[2]In 1964, the CIE defined an updated, alternate set of color matching functions using slightly different assumptions, which is what is shown in Figure 7.2.

stimulus $P(\lambda)$ in this XYZ color space (also known as CIEXYZ) is:

$$\begin{aligned} X &= k\int_0^\infty P(\lambda)\bar{x}(\lambda)d\lambda \\ Y &= k\int_0^\infty P(\lambda)\bar{y}(\lambda)d\lambda \\ Z &= k\int_0^\infty P(\lambda)\bar{z}(\lambda)d\lambda. \end{aligned} \tag{7.4}$$

The scale factor k is given by

$$k = \frac{100}{\int_0^\infty H(\lambda)\bar{y}(\lambda)d\lambda},$$

where $H(\lambda)$ is the spectrum of the white light illuminating the scene. For reflective materials with reflectivity $R(\lambda)$, use $P(\lambda) := H(\lambda)R(\lambda)$. The CIE chose the $\bar{y}$ color matching function to be identical to the human luminous efficiency function, which means that Y encodes the luminance of a color. The scale factor, k, is defined such that for an ideal diffuse reflector (the brightest and whitest color possible) Y will have a value of exactly 100. Having the values scaled to lie nominally within the range of 0–100 is mostly a convenience for human users. Many computer programs alternatively scale XYZ to lie on the range [0,1].

A natural way to discretize Equation 7.4 is to define a discrete sequence of wavelengths, $\lambda_i \equiv \lambda_0 + i\Delta\lambda$ for $0 \leq i < N-1$. Then use a simple Riemann sum to approximate the integral:

$$\begin{aligned} X &= k\sum_{i=0}^{N-1} P(\lambda_i)\bar{x}(\lambda_i)\Delta\lambda \\ Y &= k\sum_{i=0}^{N-1} P(\lambda_i)\bar{y}(\lambda_i)\Delta\lambda \\ Z &= k\sum_{i=0}^{N-1} P(\lambda_i)\bar{z}(\lambda_i)\Delta\lambda, \end{aligned} \tag{7.5}$$

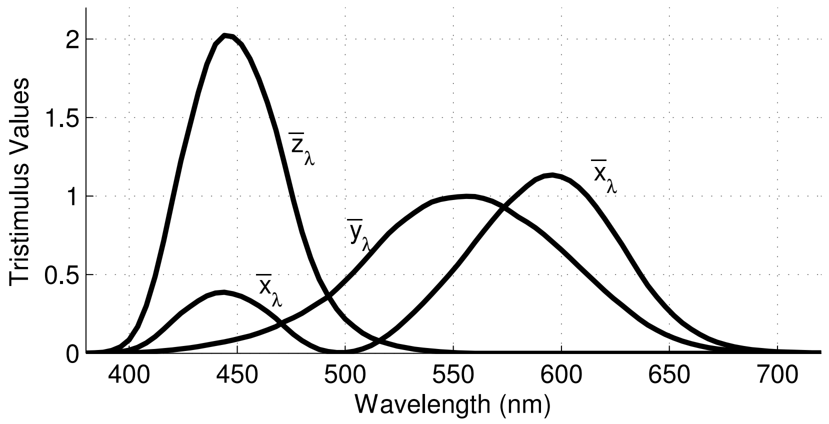


Figure 7.2: *The CIE (1964) 10-degree standard observer color matching functions used to compute XYZ color coordinates. (Data courtesy: http://cvrl.ioo.ucl.ac.uk/database/text/cmfs/ciexyz64.htm)*

where

$$k = \frac{100}{\sum_{i=0}^{N-1} H(\lambda_i)\bar{y}(\lambda_i)\Delta\lambda}.$$

There are other ways to discretize integrals besides a Riemann sum, such as Simpson's rule or Gaussian quadrature. The latter will be mentioned again later in the chapter.

Another important tool for color analysis is the chromaticity diagram. Chromaticity coordinates give hue and saturation information independent of luminance. The XYZ chromaticities are given by:

$$\begin{aligned} x &= X/(X+Y+Z) \\ y &= Y/(X+Y+Z) \\ z &= Z/(X+Y+Z). \end{aligned}$$

As can readily be seen, $x + y + z = 1$, so there are actually only two independent values. The chromaticity diagram is generally created by plotting just the first two coordinates, (x, y). By plotting the chromaticities of spectral delta functions $\delta(\lambda_0 - \lambda)$ for 380nm $< \lambda_0 <$ 700nm, one obtains the horseshoe-shaped spectral locus shown in

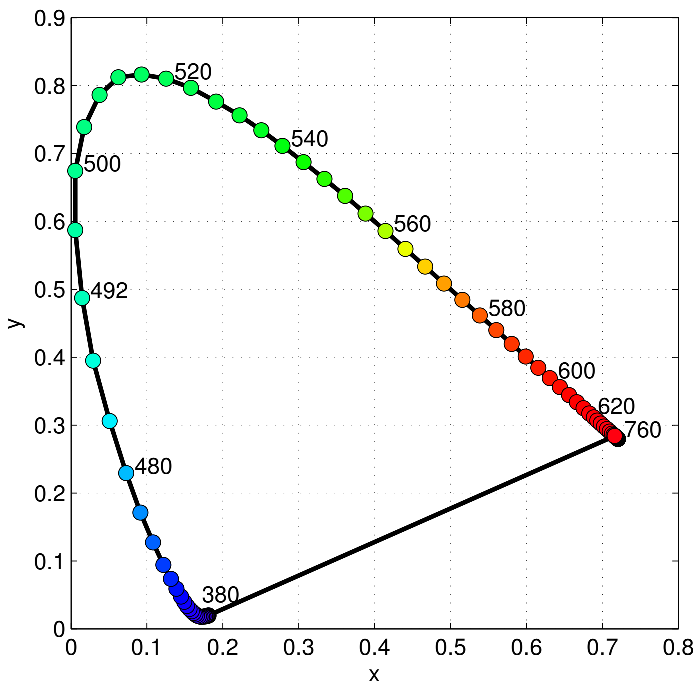


Figure 7.3: *The CIE (1964) 10-degree standard observer chromaticity diagram. (Data courtesy: http://cvrl.ioo.ucl.ac.uk/database/text/cmfs/ciexyz64.htm)*

Figure 7.3. The spectral locus contains the purest, most saturated colors possible, while the interior contains less saturated colors. Points exterior to the locus do not correspond to visible colors.

7.2.1 Metamerism

There are many spectral curves that can result in the same perceived color, as was mentioned above in the context of color matching with colored lights. The lights may combine to create a matching color without creating the exact same spectrum as the target. This phenomenon is known as *metamerism*. All that is required is that two spectra have the same integrals with respect to the eye's sensitivity functions. To achieve accurate color matching, metamerism must not be ignored. Most of the colors we see are a result of reflected light. In terms of computing XYZ coordinates, the stimulus, $P(\lambda)$ of Equation 7.4 becomes the product $R(\lambda)H(\lambda)$, of reflectance R times the light H. It is possible, then, for two materials with reflectances $R_1(\lambda)$ and $R_2(\lambda)$ to have identical XYZ coordinates under light source $H_A(\lambda)$ but to have different XYZ coordinates under light source $H_B(\lambda)$. Such metamerism can cause difficulties when one tries to match a new batch of paint to existing paint for which the formula is unknown. The match may look perfect in the lights of the paint store, but look quite different under the incandescent lights at home.

Significantly, there is no way for a 3-component color representation to capture this effect. (Johnson & Fairchild, 1999) go so far as to argue that for color-sensitive applications, a full-spectrum color model is necessary. I propose a practical compromise later in this chapter to balance between the need for full-spectrum data and the need for efficiency in a real-time rendering system.

7.2.2 Color Space Transformations

Given two tristimulus color spaces defined in terms of color matching functions, there exists a simple 3×3 transformation matrix that will convert between the two spaces. For instance, there is a 3×3 matrix that will convert from XYZ coordinates to RGB coordinates. This is the transformation necessary in order to display XYZ colors on a monitor. However, the RGB color resulting from the transformation may be invalid. If the color is not in the gamut of the target color space, one or more of the components will be negative. Another issue is luminance overflow. The magnitude of the color in one or more channels may be greater than the maximum reproducible luminance of the display device.

The problem of displaying such out-of-gamut colors is an important issue for a realistic painting system, since many commonly used real-world pigments fall outside the gamut of existing monitors. Therefore, a gamut mapping algorithm is required. Simple solutions such as clamping color values to $[0, 1]$ generally perform poorly. Such methods can change hues and create noticeable discontinuities in smooth regions of color. Gamut mapping is an active area of current research in graphics. I revisit this topic later in the chapter.

7.3 Overview of Color Mixing Models

I have already referred to additive color mixing in the discussion of color matching above. But there are several different models for how colors mix in the real world. Not every type of color or colorant in the real world mixes in the simple linear way lights do. Mixing paints to create new colors is a fundamental operation in painting, so it is important to have a reasonable mathematical model for mixing. (Fishkin, 1983) describes four mixing models, additive, averaging, subtractive, and pigmented, which I summarize here.

7.3.1 Additive Mixing

As already mentioned, colored lights mix *additively.* Given several lights described by RGB intensities $C_i = (R_i, G_i, B_i)$, if all lights illuminate the same diffuse white surface, the resulting color will be:

$$C = \sum_i C_i.$$

The light intensities are simply added together. With additive mixing, red light $(1, 0, 0)$ mixed with green light $(0, 1, 0)$ results in yellow light $(1, 1, 0)$. When two colors are mixed additively, the chromaticity of the result always lies on the straight line segment joining the chromaticities of the two source colors on the chromaticity diagram. When three colors are involved the result always lies in the triangle defined by the three source colors, and in general when N colors are combined, the resulting chromaticity is always a convex combination of the N chromaticities, and thus lies within the N-gon convex hull of the N colors. The chromaticity diagram makes it immediately clear why mixtures of three visible light sources can never reproduce all visible colors—it is geometrically impossible for a triangle *inscribed* within the spectral locus shown in Figure 7.3 to simultaneously *circumscribe* it.

7.3.2 Average Mixing

Suppose now that instead of both lights illuminating the same area, the areas illuminated by each light are interleaved such that every point on a surface is only illuminated by one light or the other. If each light covers exactly half the total area, it is clear that the intensity (flux per unit area) from each light leaving the surface is only half as great as it would be otherwise. Assuming that the interleaving of the light areas is at a scale below the eye's resolving power, then the apparent color will be the *average* of the two light colors. If several lights mix together in this way, each covering a fraction f_i of the surface, then the color will be

$$C = \sum_i f_i C_i$$

where $\sum_i f_i = 1$. More practically, this is the same type of mixing achieved when a half-toning printing process is used. It is also the type of mixing used in the dAb system of Chapter 4, so one could say that the dAb model treats two paints mixed together as if their relative volumes represent areal coverage.

7.3.3 Subtractive Mixing

In *subtractive* mixing each colorant removes light of a particular color. Consider the case of dyes, which are a common example of a subtractive colorant. Each dye has a transmittivity (or reflectivity) that can be described in terms of three components, $T_i = (R_i, G_i, B_i)$, with each component value between zero and unity. The light transmitted through (or reflected off) the surface is the product of the light color and T_i. The combination of several dyes yields a mixture whose transmittivity(reflectivity) is the product of the constituents[3]:

$$T = \prod_i T_i.$$

With subtractive mixing, fully saturated cyan $(0, 1, 1)$ mixed with saturated yellow $(1, 1, 0)$ results in saturated green $(0, 1, 0)$.

In actuality, using just three wavelengths is an approximation to the true physical behavior. The actual result is given by the product of full-spectrum wavelength-dependent transmittance(reflectance) functions $T_i(\lambda)$. In other words:

$$T(\lambda) = \prod_i T_i(\lambda).$$

Then the resulting RGB color can be obtained by integrating against the RGB color matching functions, or XYZ color matching functions followed by an XYZ to RGB linear transformation. It should be clear mathematically that in general for a light spectrum H, transmittances T_1 and T_2, and one of the color matching functions like $\bar{r}$:

$$\begin{aligned} & \int_0^\infty H(\lambda)T_1(\lambda)T_2(\lambda)\bar{r}(\lambda)d\lambda \\ \neq \; & \left(\int_0^\infty H(\lambda)\bar{r}(\lambda)d\lambda\right) \cdot \left(\int_0^\infty T_1(\lambda)\bar{r}(\lambda)d\lambda\right) \cdot \left(\int_0^\infty T_2(\lambda)\bar{r}(\lambda)d\lambda\right). \end{aligned}$$

Thus the result of subtractive mixture in terms of RGB components is not the same as that performed across the full spectrum. Mixtures of subtractive colorants generally fall on curved segments in the chromaticity diagram.

Dyes, colored filters, photographic emulsions, and other materials that mix subtrac-

[3]Thus, *multiplicative mixing* would probably be a more apropos name, but the term subtractive color has stuck.

tively, selectively absorb light but do not cause any significant scattering. The amount of light absorbed depends on the concentration of the dye, c, the thickness of the layer of dye, d, and the per-wavelength absorption coefficient of the dye, $K(\lambda)$. Given these parameters, the transmittance, $T(\lambda)$, through a dyed layer of material, like a filter, is accurately modeled by Beer's Law over a wide range of parameter values:

$$T(\lambda) = e^{-cK(\lambda)d},$$

or, equivalently,

$$\ln(T(\lambda)) = -cK(\lambda)d.$$

For combinations of dyes one uses the product of the transmittance of individual layers:

$$T(\lambda) = \prod_i e^{-c_i K_i(\lambda) d_i}.$$

So if one is working in terms of logarithms the exponents are simply added together (or subtracted, since they are negative):

$$\ln(T(\lambda)) = \sum_i -c_i K_i(\lambda) d_i,$$

which is possibly where the term "subtractive" originates (Poynton, 2002).

7.3.4 Pigment Mixing

Finally, some materials such as paints, plastics, and textile colorants contain pigment particles, which both scatter and absorb light. The difficulty in modeling the interaction of light with such a material is that light can enter the material and be scattered and reflected and refracted internally multiple times before ultimately re-emerging or being absorbed. A few possible light paths are shown in Figure 7.4. There are several

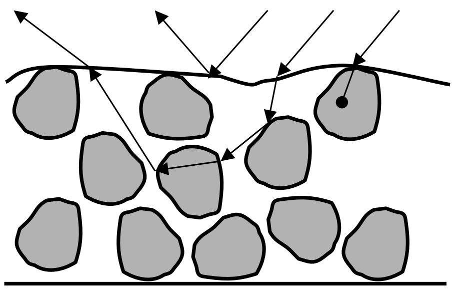


Figure 7.4: *Paint consists of pigment particles suspended in an optically neutral medium. Light can take many paths on its way through the paint. It can be simply reflected off the surface according to Fresnel reflectance; it can enter into the medium and scatter multiple times before exiting; or any time it encounters a pigment particle it could be absorbed.*

approaches to dealing with the complexity induced by this multiple scattering. The Monte Carlo approach, used frequently in nuclear physics, is to actually trace paths of many individual photons through the material and develop a probabilistic model of the distribution of light energy. Another approach is to assume the effect is isotropic and use diffusion models as in the BSSRDF model (Jensen et al., 2001).

Another approach was presented by the German scientists Paul Kubelka and Franz Munk in 1931 in the form of a simple set of differential equations to describe the transport of light in pigmented materials (Kubelka & Munk, 1931). The model is based on the assumption of a homogeneous layer of medium that is infinite in extent. By symmetry, then, all lateral flux can be ignored, since it will be exactly balanced out by an equal and opposite flux. Thus the model examines just the flux balance in 1D, perpendicular to the surface. The model describes the material properties in terms of just two wavelength-dependent parameters: an absorption constant, $K(\lambda)$, and a scattering constant, $S(\lambda)$. The resulting analytical solutions to Kubelka and Munk's differential equation have found wide scientific utility in areas as diverse as the study of paint, paper, textiles, and skin, as well as in art conservation and planetary science.

They have also been used in computer graphics to a limited extent (Haase & Meyer, 1992; Curtis et al., 1997; Rudolf et al., 2003; Baxter et al., 2004b).

Kubelka extended the 1931 work in two subsequent articles. He first presented simpler hyperbolic solutions to the the differential equation of transport presented in original article, allowing for simpler computation of the reflectance and transmittance of semi-opaque layers (Kubelka, 1948). Later he presented a simple method for computing the reflectance and transmittance of several different layers composited on top of one another(Kubelka, 1954). I use both of these results in the real time paint-rendering system presented in this chapter (middle part of Equation 7.6, and Equation 7.11). Additionally I also make use of the work of (Duncan, 1940; Duncan, 1962) who showed that the absorption and scattering coefficients for mixtures of paints can be obtained by the sum of the fractional amounts of absorption and scattering from each pigment present (right hand side of Equation 7.6). In other words the scattering and absorption coefficients of a mixture are a convex combination of the scattering and absorption coefficients of the constituent pigments.

7.4 Previous Work

There is a large body of previous work spread across several areas that is relevant to this chapter. Researchers have been studying ways to represent, manage, and reproduce color on the computer for a long time. I have already given an overview of much of this background material in the previous sections of this chapter. Another category is painting programs, which have been rendering paint one way or another since their inception. A third category is previous work related to the the Kubelka-Munk equations.

Alvy Ray Smith's original "Paint" program (Smith, 1978) offered perhaps one of the first 2D methods for simulating the look of painting. A paint rendering model that offers the look of thick, viscous paint with bump-mapping can be found in (Cockshott et al., 1992). Cockshott's system performs color mixing in HSV space. Corel Painter (Painter 8, 2003) is a commercial product that features a variety of digital natural media. Painter seems to use an additive or averaging color model most of the time, but it provides a mixing palette that appears to use a subtractive, CMY dye model for color mixing rather than an additive, RGB color model. This is probably the method discussed in (Zimmer, 1994). Painter also associates heightfields with the canvas and renders this as bumpiness. Most painting programs that perform color mixing use an additive or averaging model.

(Oddy & Willis, 1991) presents a "physically-based" color model that incorporates a variety of blending modes intended to mimic the behavior of real-world filters and pigments. The "beta channel" is proposed as a generalization of the alpha channel to recreate various effects, and three additional channels of color are added to represent the medium color of a material separate from its pigment color. Though the model aims to mathematically model the differing optical behaviors of pigments and filters, it is really more of a mathematical abstraction than an accurate model of the physical optical behavior of materials.

The Kubelka-Munk equations have been used in a variety of rendering contexts in computer graphics. (Haase & Meyer, 1992) demonstrated the utility of the K-M equations for rendering and color mixing in both interactive and offline applications, including a simple "airbrush" painting tool. (Dorsey & Hanrahan, 1996) used K-M layer compositing to accurately model the appearance metallic patinas. (Curtis et al., 1997) used the K-M equations for optically compositing thin glazes of paint in their watercolor simulation, and (Rudolf et al., 2003) used the same form in their wax crayon simulation. None of these implementations offers the true real-time rendering desired in an interactive application, however.

Outside of computer graphics, Kubelka-Munk theory has had widespread application. Scientists interested in industrial uses of paint have long used the Kubelka-Munk equations as a tool, see e.g. (Callet, 1996) for a recent example. In the field of planetology, remote measurements of reflectance properties can tell a scientist a great deal about the composition of a planet's surface (Hapke, 1993), and for that reason the Kubelka-Munk diffuse reflectance model has proven useful to planetologists. (Johnston-Feller, 2001) describes how K-M theory can be used in art conservation and the restoration of museum artifacts like paintings. Johnston-Feller describes how K-M theory can be used as a tool both to non-destructively identify pigments and also to calculate pigment concentrations for color matching. Recently Poirer presented a model for the coloration of human skin based on the Kubelka-Munk equations (Poirier, 2004). General references on reflectance spectroscopy like (Körtum, 1969) and on color science like (Judd & Wyszecki, 1963) list many more references to applications of K-M theory.

7.4.1 Color Samples for Kubelka Munk

As mentioned, the Kubelka-Munk (K-M) model has been used previously in graphics in order to render pigmented materials that exhibit subsurface scattering and absorption. Although both (Curtis et al., 1997) and (Rudolf et al., 2003) use K-M to simulate artistic media, they both used simpler rendering during user interaction, and then added the more accurate colors as a post processing step. (Haase & Meyer, 1992) used a custom four-wavelength representation of the K-M parameters based on Meyer's previous work (Meyer, 1988), while the others worked in standard three-wavelength RGB space.

Meyer's four-wavelength color encoding was developed to be both more accurate than RGB and still efficient enough for execution on the computers available when it was written (Meyer, 1988). This encoding was based on integrating against the human visual response functions in ACC color space. He used Gaussian quadrature in order to find four abscissas wavelengths that when integrated against, would reduce color error compared to standard RGB models. However, Johnson and Fairchild point out that under some lighting conditions, any trichromatic color space such ACC will give incorrect results. They suggest using full-spectral color representations and present a real-time full-spectral rendering system capable of per-vertex diffuse lighting (Johnson & Fairchild, 1999).

In order for artistic media to be properly simulated, colors must blend properly. Furthermore, artists must see the results of their actions continuously in order to react to the new output and produce painterly works. Also, an artist may desire to preview how colors will appear under different lighting conditions. In order to satisfy all of these conditions, I use a novel eight-component color representation that combines Meyer's use of Gaussian quadrature, Johnson's use of full spectral data, and Kubelka-Munk color mixing.

7.5 Measuring the paints

In order to obtain realistic parameters corresponding to real paint media, I measured several standard oil paint colors that are common to an artist's palette. See (Gair, 1997), for example, for a list of standard oil pigments. Samples of each paint were applied to acetate sheets and flattened with a palette knife to obtain a smooth surface. Each sample was thick enough to achieve complete hiding (i.e. zero transmittance of light through the layer). Different mixtures of the paints with each other in measured ratios were also prepared. The light energy reflected off of the samples was then measured using Photo Research's Spectra Scan PR-715, a spectroradiometer. In this manner, 101 reflected energy values in the visible spectrum (380-780nm) with 4nm spacing were obtained for each sample. The measurement rig was set up as in Figure 7.5, with the light source at approximately a 60-degree angle from the sample's normal. The angle was chosen first to minimize the amount of specular reflection measured, and, second, because this geometry is significant in that the assumptions of K-M theory are only exactly satisfied for incident radiance that is either uniform and isotropic *or* in parallel rays at a 60 degree angle (Kubelka, 1948). I used a reflectance standard made of Fluorilon FW in order to measure the output of the light, and a neutral density filter to attenuate the light reflected by the standard to a level measurable by the PR-715. The attenuation of light intensity by the neutral density filter was determined by taking two measurements of a darker paint sample that was in a range to be measured both with and without the filter. Given the measurements with and without filter, the effect of the filter could be determined, and was then factored out of the light measurement to obtain the true light spectrum. Finally the curves giving light energy reflected off the samples were divided through by the the energy reflected off the diffuse reflectance standard. The resulting curves represent the reflectances of the paint samples, the fraction of incoming light per wavelength diffusely reflected.

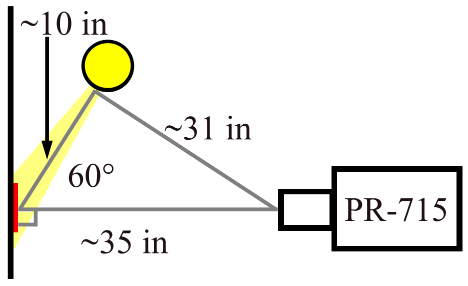


Figure 7.5: *Our setup for measuring paint reflectances. Each paint was placed in turn at the target location, indicated in red. Several measurements were made per paint sample and averaged.*

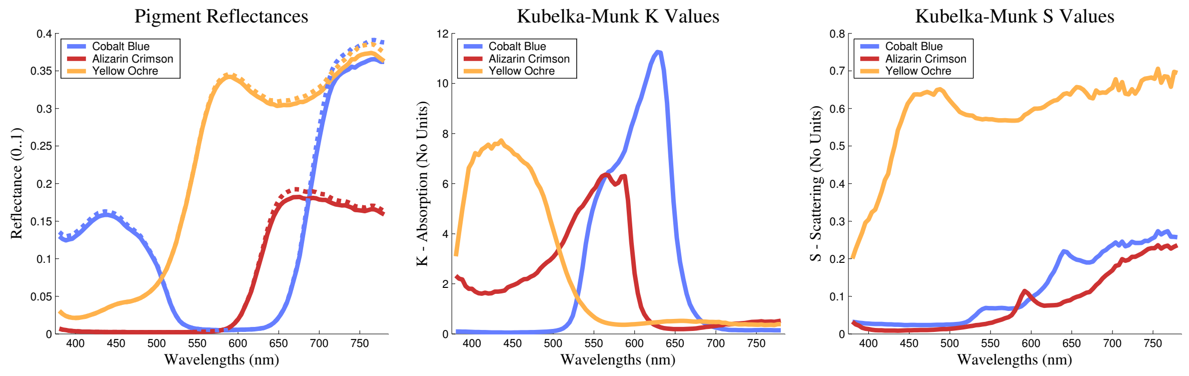


Figure 7.6: *Some results from measuring real oil paints. The left graph shows the measured reflectances after factoring out the spectrum of the incident light source (dotted lines), and the computed reflectances after solving for K and S values (solid lines). The right two graphs show the Kubelka-Munk absorption (K) and scattering (S) coefficients computed from the measured reflectance data.*

7.6 Converting to Kubelka-Munk

Using the reflectance curves computed from the paint measurements, I calculated the K-M absorption and scattering (K and S) coefficients for each paint sample at each wavelength. Given mixtures $1 < i < M$ of the pure pigments $1 < j < N$, one can use the following equation taken from K-M theory (Kubelka & Munk, 1931; Duncan, 1962; Haase & Meyer, 1992):

$$\left(\frac{K}{S}\right)_{\text{mix},i} = \frac{\sum_j K_j c_{ij}}{\sum_j S_j c_{ij}} = \frac{(1 - R_{\infty,i})^2}{2R_{\infty,i}}. \tag{7.6}$$

This relates the reflectance of mixture i, $R_{\infty,i}$, to the absorption and scattering values of each constituent pigment, K_j and S_j, and their relative concentrations, c_{ij}. Pigments not involved in a particular mixture are assigned zero concentration. The dependence on wavelength is assumed in the above equation and those that follow; for brevity the λ will be omitted.

From Equation 7.6 the next step is to assemble a linear system (for each wavelength) of the form

$$\mathbf{A}\begin{pmatrix}\mathbf{K}\\ \mathbf{S}\end{pmatrix} = \begin{pmatrix}\mathbf{C} & -\mathbf{Q}_R\mathbf{C}\end{pmatrix}\begin{pmatrix}\mathbf{K}\\ \mathbf{S}\end{pmatrix} = \mathbf{0} \tag{7.7}$$

where $\mathbf{C} = \{c_{ij}\}$ is an $M \times N$ matrix containing the paint concentrations, and $\mathbf{Q}_R$ is an $M \times M$ diagonal matrix, containing the right-hand sides of Equation 7.6 along the diagonal. The unknowns, $\mathbf{K}$ and $\mathbf{S}$, are both $N \times 1$ vectors.

In general, for $M > 2N$, the zero vector is the only solution, since the equations will have zero nullity (full column rank). Instead, what is desired is a least-squares solution that minimizes $A^T A$, subject to a constraint that enforces non-triviality of the solution. This can be enforced using a simple equality constraint on one of the variables, say $S_k = 1$ for some k. It is further required that each K_j and S_j be positive. Together these requirements specify a simple quadratic program (QP) that can be solved using a standard QP solver. I made $M = 71$ measurements of different mixtures involving $N = 11$ different paints, including the N measurements of the pure pigments alone. I chose to enforce $S_k = 1$ for k corresponding to Titanium White. Note that regardless of how the equations are solved, it is always necessary to choose some value arbitrarily, since K and S always appear in ratio. Figure 7.6 shows the measured reflectances, computed K and S values, and reflectance computed from those K and S values for three of the paint samples, calculated as just described.

7.7 Lights, Sampling and Gaussian Quadrature

Color should be treated as accurately as possible, but it is not feasible to store the full 101-wavelength K and S samples on a per-pixel basis or to compute the K-M model per-pixel on all of this data in an interactive system. Both the memory required to store 202 coefficients at every pixel, as well as the computation required to convert these values to RGB would be prohibitive in a real-time system. In fact, the fragment program would even exceed the allowable instruction length of some GPUs.

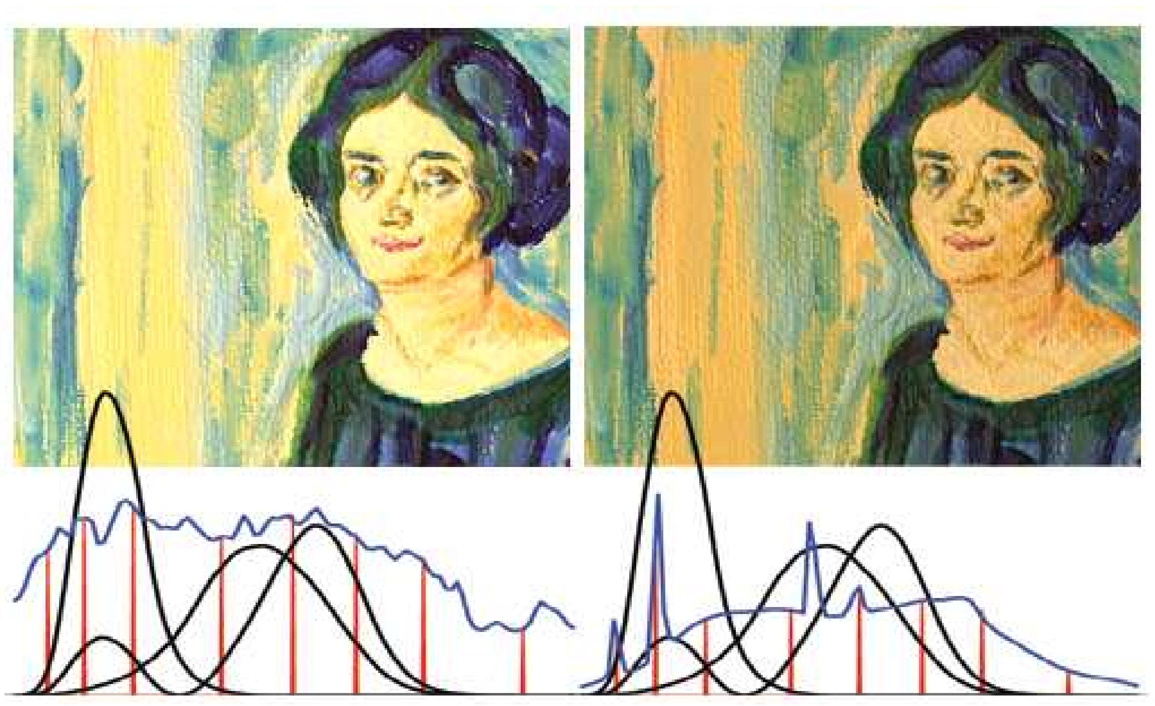


Figure 7.7: *A comparison of the same painting created in IMPaSTo under two different light sources. On the left, the painting is illuminated by a 5600K bulb. On the right, it is illuminated under CIE Fluorescent Illuminant F8. Graphs of the light spectra are in blue, the 8 sample wavelengths chosen by IMPaSTo in red, and the CIE XYZ integrating functions are shown in black for reference.*

I thus turn to numerical integration for a way to reduce the amount of wavelength data required in per-pixel calculations. Fortunately, most naturally-occurring reflectance spectra, including those of common paint pigments, are fairly smooth functions, and are thus well approximated by polynomials of moderate degree. I take advantage of this by using a Gaussian quadrature numerical integration scheme (Warnick, 2001) to compute the final conversion of per-wavelength K-M diffuse reflectances to RGB for display.

My system stores the original 101 K and S samples for all paints, energy spectra for the lights, and base reflectances for the canvas and palette. Upon choosing one specific light spectrum (e.g., the CIE Standard Illuminant D65), an automated Gaussian quadrature engine finds eight sample wavelengths and weights using a weighting function based on the XYZ color-matching functions combined with the light's energy spectrum. Specifically the XYZ weighting function used is simply the sum of the three XYZ color-matching curves. This ensures that more importance is given to the regions of the spectrum that have the most influence on the overall XYZ coordinates. After the eight sample wavelengths are chosen, each of the complete spectra are reduced to just these chosen wavelengths (See Fig. 7.7).

Since the weighting function is guaranteed to be nonnegative, Gaussian quadrature will return sample wavelengths and weights internal to the integrating region. In this way, wavelengths are chosen that are influential to both the final integration function (based on the human visual system) and the lighting environment. For instance, if a bluish light is selected to illuminate the canvas, the system will choose eight wavelengths that are biased more towards the blue end of the spectrum.

There are several reasons to settle upon eight as the number of sample wavelengths. The goal is to maximize accuracy while minimizing memory usage and computational cost. Duncan has conducted studies of the accuracy of the Kubelka-Munk equations and recommends a minimum of seven samples for reliable color prediction, and ideally twice as many samples if feasible (Duncan, 1949). His work did not use a quadrature technique for integration, however, so it is likely that using eight samples with Gaussian quadrature is already more accurate than eight samples with a more naïve integration technique. In terms of speed and efficiency, the vector nature of GPU fragment processors makes for efficient handling of the eight samples. Many GPU operations can be carried out in parallel on four components at a time, making eight components only slightly more computationally expensive overall than 5,6,or 7 components. For memory utilization, the eight samples can be stored compactly in either two textures or in one floating point texture packed as half-precision floating point.

7.8 Rendering Pipeline and GPU Implementation

I use fragment shaders written in NVIDIA's Cg programming language to calculate the overall RGB reflectance of the painted canvas, the palette, and the brush bristles. As was shown in Table 6.2, I use two textures that, with their eight channels, represent the concentrations of the eight pigments simultaneously allowed at any pixel, and then another texture as the thickness for the paint on that texel in that layer.

A multi-pass approach allows for several layers of pigment to be stacked on top of each other. The rendering pipeline closely follows the stages of Figure 7.8. Each stage is implemented as a separate fragment program. The first three fragment programs

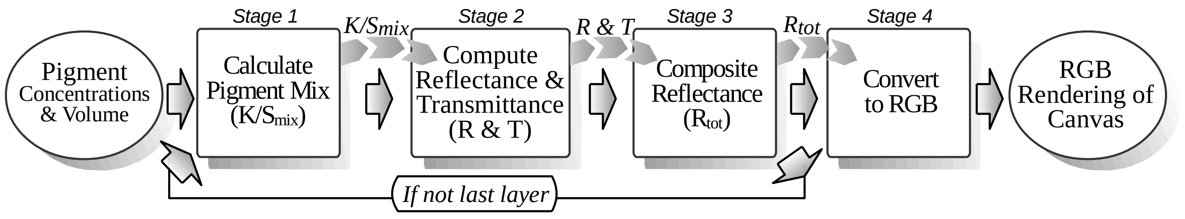

Stage 1
Stage 2
Stage 3
Stage 4
Pigment Concentrations & Volume
Calculate Pigment Mix (K/Smix)
K/Smix
Compute Reflectance & Transmittance (R & T)
R & T
Composite Reflectance (Rtot)
Rtot
Convert to RGB
RGB Rendering of Canvas
If not last layer

Figure 7.8: *Steps in Kubelka-Munk rendering. I begin with per-layer pigment concentrations and volume (per-pixel thickness). Stage 1 computes the absorption and scattering coefficients (K/Smix) for a layer of paint using Equation 7.6, then Stage 2 uses this to compute the reflectance and transmittance of that layer, according to Equations 7.8–7.10. Stage 3 uses Equation 7.11 to composite the single layer's reflectance with the reflectance of all layers beneath it. I repeat Stages 1–3 for each layer from bottom to top, and finally in Stage 4 I convert the result to RGB according to the current light spectrum and other lighting parameters.*

calculate the final reflectance of any one layer of paint. Stage 1 calculates K/S and S for the pixel by using Equation 7.6, which maps well to graphics hardware (dot products and 4 channel adding). Stage 2 calculates the reflectance and transmittance for this one layer using the following (Kubelka, 1948; Curtis et al., 1997):

$$\begin{aligned} b &= \sqrt{(K/S)(K/S+2)} && (7.8) \\ R &= \frac{1}{1+K/S+b\coth(bSd)} && (7.9) \\ T &= bR\,\operatorname{csch}(bSd). && (7.10) \end{aligned}$$

where d represents the thickness of one layer of paint. Stage 3 calculates the reflectance of this layer composited on top of the previous layers using

$$R_{\text{tot}} = R + \frac{T^2 R_{\text{prev}}}{1 - RR_{\text{prev}}}. \tag{7.11}$$

These three stages use the eight wavelengths chosen via Gaussian quadrature, and are iterated over once for each layer of pigment. In my implementation, only the wet paint is represented as pigments in the runtime data, requiring these multi-pass iterations. As paint dries, its reflectance is calculated and added into the base canvas. To change the light spectrum when dry layers are present, I do as many passes as are necessary in order to "bake" the dry paints into the base canvas' reflectance, and then need only perform these calculations once for the wet paints. Stage 4 uses the weights derived via Gaussian quadrature and the XYZ integrating functions in order to transform these 8 wavelength values first into XYZ space and then via a 3×3 matrix into RGB space for display.

7.8.1 XYZ to RGB conversion

As mentioned in Section 7.2.2 some XYZ colors can lead to out-of-gamut values when transformed to RGB space. The two cases to deal with are negative values, which are undisplayable hues (i.e. more saturated than is possible to achieve on the display device), and luminance overflow, which is the case with colors too luminous to display.

Gamut mapping techniques can be either *local* or *global*. Local methods consider the value of each color to be transformed independently, whereas global methods consider the actual gamut spanned by all of the colors in an image. I have chosen to use local methods simply because they are much more computationally efficient.

For negative color components, (Fishkin, 1983) recommends a clamping procedure in RGB chromaticity space that preserves the hue of the color, but reduces its saturation to a displayable level. RGB chromaticities are given by:

$$\begin{aligned} r &= R/(R+G+B) \\ g &= G/(R+G+B) \\ b &= B/(R+G+B), \end{aligned}$$

and just as with XYZ chromaticities, $r + g + b = 1$. The chromaticity of any neutral color (any shade of gray) is therefore $(1/3, 1/3, 1/3)$. The basic idea is to clip the line segment that connects the out-of-gamut chromaticity value, (r, g, b), and white to the triangular gamut of displayable colors. The pseudocode below is a refined version of that presented by Fishkin. The key observation is that the most negative component is the one that determines the factor t by which the segment's length must be reduced, and this can be determined simply by looking at the chromaticities' projection on that axis alone, which means one short function can handle intersections with any of the three edges of the chromaticity triangle:

```
float3 clampRGBChromaticities(float3 RGB)
        float minc = min(RGB.r, RGB.g, RGB.b)
        if (minc < 0)
                float sum = RGB.r + RGB.g + RGB.b
                float3 rgb = RGB/sum
                float t = (1/3)/(1/3 - minc)
                float3 white = (1/3,1/3,1/3)
                rgb = white + t*(rgb-white)
                RGB = rgb * sum
        endif
        return RGB
```

This procedure works well, but exhibits some issues common to most local gamut mapping methods. For one, the onset of clipping is discontinuous, which can be noticeable if it occurs in the middle of a gentle color gradient. For luminance overflow, I use simple clipping to the RGB unit cube, though, in practice, overflow is best prevented by appropriate setting of the light intensity.

Finally, no discussion of color space transformation today would be complete without mention of the International Color Consortium (ICC). The ICC was formed in 1993 by eight industry vendors in order to develop and promote color interchange standards. An ICC compliant color management system (CMS) uses vendor-provided ICC color profiles to manage the transformation between various device color spaces and the intermediate ICC "profile connection space" (PCS). The PCS is just a device-independent color space based on either CIEXYZ or CIELAB with a CIE D50 white point. The PCS also specifies the number of bits to be used in encoding colors along with other implementation details that are not specified by the CIE. The ICC defines four rendering intents for gamut mapping: relative colorimetric, absolute colorimetric, perceptual, and saturation. Each preserves different aspects of the source color space when mapping it to the destination gamut. These are intents, however, not actual algorithms. The actual implementation depends on the vendor of the CMS, and was intentionally left open to allow for future algorithms. An ICC profile for an output device, and for a specific rendering intent, contains essentially a large lookup table, mapping colors in the PCS to colors in the device space. One such lookup table must be specified for each rendering intent on each device. More information about the ICC, including technical details of the latest ICC profile specification, ICC.1:2003-09 (version 4.1.0), can be found at http://www.color.org.

It would be desirable to make use of the ICC profiles in the final XYZ to RGB stage of my paint rendering algorithm. However, this is not as straightforward as it might seem, since profiles are designed more or less as black boxes that are accessed via the color management system by requesting transformation of individual colors. Such a system does not seem well suited for use in real-time rendering with programmable shaders. It is, however, an interesting area for future work.

7.8.2 Glossy and Specular Reflection

The K-M equations assume no interface between the pigmented material and surrounding environment (typically air), and thus do not model glossy surface reflection, just diffuse reflection. In a real paint sample, some fraction of the incoming light is immediately reflected off the surface. This phenomenon is due to the differing indexes of refraction between the environment and the medium in which pigments are suspended (e.g. linseed oil). The Fresnel equations give the reflectance ratio, ρ, based on the two indexes of refraction, n_1 and n_2, where n_2 is the optically denser medium. For light perpendicular to the interface, the Fresnel equation reduces to:

$$\rho = \left(\frac{n_2 - n_1}{n_2 + n_1} \right)^2 .$$

For air ($n_1 = 1.0003$) and linseed oil ($n_2 = 1.5$), ρ is about 4%. Light reflected off the surface of a dielectric (non-metal) in this way is not influenced by the color of the material, meaning glossy reflections and specular highlights have color of the light, not the surface. Since the K-M calculations only give the diffuse reflectance, I complete the lighting computation with per-pixel dot-product bump-mapping and specular highlights

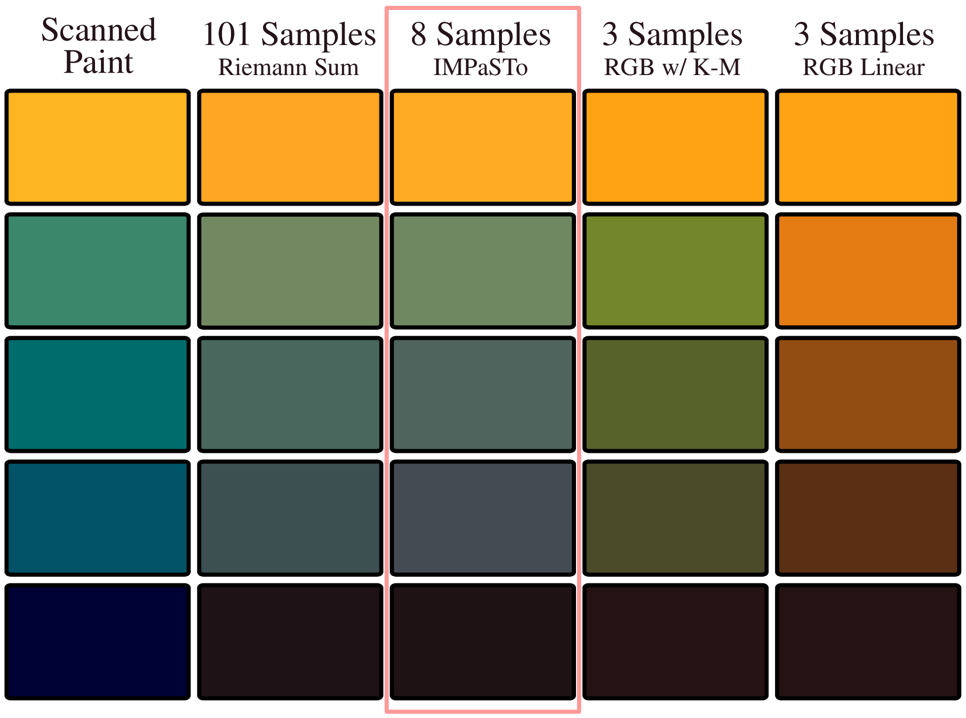


Figure 7.9: *The left column shows graded mixtures of Yellow Ochre and Prussian Blue under a 5600K light. The right four columns show computer simulations of the mixtures using different numbers of sample wavelengths and differing techniques. As can be seen, linear RGB blending wrongly predicts brown. Although my implementation of K-M blending does not match the scanned colors exactly, the important feature to note is that the result of using my 8-sample Gaussian quadrature is almost identical to that using 101 samples. Thus, given more accurate reflectance data as initial input, I should be able to match the real samples very closely.*

computed with the Blinn-Phong model. The specular contribution is computed in RGB space, since in RGB space the specular color is simply the color of the light, which by definition is white.

7.9 Color Comparison

Figure 7.9 shows the results of the rendering algorithm and comparisons of the blending results for my method and several alternatives. On the left, the figure shows a scanned sample of actual paint mixtures. The next column computes the color of mixtures using the Kubelka-Munk equations at 101 wavelengths to obtain the spectral reflectance at 101 wavelength samples. Then reflectance is then converted to XYZ using Equation 7.5, with $H(\lambda_i)$ set to a particular light source spectrum. The third column is obtained in a similar manner, but using only 8 sample wavelengths, and Gaussian quadrature to estimate the XYZ integrals. The fourth column uses Kubelka-Munk as previous authors have done, with just three wavelengths treated directly as the R, G, and B reflectances for display, bypassing the integration step. Finally the rightmost column just shows blending performed with linear interpolation of the R, G, and B color components. Some difference between the scanned paints and the computed colors is to be expected because the light source used by the flatbed scanner was not the same as that used in the color calculations. However, note that the 8-wavelength Gaussian quadrature method I propose achieves nearly the same results as using all 101 wavelength samples, at greatly reduced runtime computational cost. Also note that the results when using fewer samples, as previous researchers have, are noticeably different.

7.10 Results

I have coupled this paint rendering model with the paint dynamics models in the previous two chapters (Stokes Paint and IMPaSTo), and tested the implementation on a variety of systems, equipped with NVIDIA GeForceFX 5900 Ultra, GeForce 5950, and GeForce 6800 Ultra graphics cards. The re-rendering of the paint tiles is performed lazily and only after the brush modifies a given tile. Because of this localization and lazy updating, rendering has not been a bottleneck thus far, but rather the paint dynamics simulation.

The paintings in Chapters 5 and 6 were all rendered using the real-time interactive eight-sample Kubelka-Munk paint appearance model presented in this chapter.

7.11 Limitations and Future Work

The astonishing aspect of K-M theory is not the accuracy with which it predicts the colors of pigmented mixtures, but rather the fact that it can predict real-world color mixtures as all, given the routine violation of so many of the primary assumptions underlying the theory. Some of those generally violated assumptions:

- *Perfectly diffuse lighting (or parallel lighting at exactly 60°).* My implementation allows the user to move the light source anywhere, and the colors still seem plausible.

- *Perfectly random distribution of particle orientations.* Most paints are fairly random, but metallic flake paints are a notable exception. Of course, metallic flakes also violate other assumptions.

- *Perfectly homogeneous materials.* This is violated anywhere two different concentrations, or thicknesses, of pigments fall side-by-side.

- *Perfectly smooth surfaces.* I allow the surface to have bumpy geometry that technically should cause a significant change due to the angles of incidence and refraction at entry.

- *Infinite extent of the sample.* Clearly one is limited to a finite canvas given finite computer memory.

- *Perfectly isotropic scattering.* (Johnston-Feller, 2001, p.119) discusses the breakdown of this assumption when either the concentration or layer thickness becomes very small. In that case forward scattering dominates, violating the assumption of an "intensely light-scattering material" (Kubelka, 1948).

Despite all of these violated assumptions, the results are still quite realistic the majority of the time.

In terms of future improvements to the algorithm, several researchers specializing in other fields have proposed improvements to the Kubelka-Munk model, such as (Duntley, 1942) and (Callet, 1996). Recently there has been some progress in rendering techniques for semi-translucent materials with subsurface scattering using diffusion models BSSRDF (Jensen et al., 2001). This work seems relevant to paint rendering as well, since it also involves subsurface scattering. Medical researchers have also proposed some improved models for light transport through skin such as (Poirier, 2004).

Currently the paint rendering algorithm does not incorporate a notion of varying medium or of liquid content. An interesting area for future improvements would be to model different media with different indexes of refraction. However, the Kubelka-Munk coefficients are particular to the media used when making the measurements (Fishkin, 1983), so providing a realistic optical model for different media is non-trivial. Also, as the ratio of pigment volume to media volume increases it can give the paint a more matte appearance. I currently do not model this effect, nor the effect of drying on the surface appearance. Another related area for possible rendering improvements would be to add more sophisticated surface reflectance models, such as allowing an arbitrary BRDF.

As already mentioned, improved gamut mapping and integration with the standardized ICC color management systems, now included as a component of most desktop windowing systems, is also very desirable.

7.12 Summary

This chapter has presented an accurate, real-time color mixing and layer compositing algorithm based on the Kubelka-Munk equations and Gaussian quadrature integration that executes efficiently using the programmable shading available on modern graphics hardware. Compared with previous paint rendering systems, my paint rendering technique offers both greater interactivity and greater color accuracy.

Chapter 8

Computer Interface

Despite its usefulness, the computer is no panacea for dispensing with the hard work and diligent imagination requisite to any serious art. It is no elixir for the creative agony often unavoidable in the process of giving birth to new expressions
— Timothy Binkley in the introduction to (Schwartz, 1992)

The interface for a realistic computer painting system is an important consideration. The high-fidelity simulation of paint enables the possibility of providing a useful tool to those more interested in traditional painting than computer painting. But care must be taken in designing the interface for the system, so as not to alienate that very user group by overwhelming them with a profusion of buttons and complex controls. However, no matter how capable the interface, the quote above brings to attention the fact that creating serious art requires serious effort, no matter what the medium. The interface should assist the painter, and complement the painter's abilities, but it is no substitute for the painter's artistic vision and talent.

An interface is desired that will be easy to learn and easy to use, while at the same time maximize the potential for skill transfer, and enable expert users to work efficiently. One important and novel aspect of my system is the haptic response of the system to user input.

In this chapter I discuss both the overall interface design and interface components as a whole, as well the specifics behind my haptic feedback algorithms.

Part I: Painting Interface

In the painting system interface, I have attempted to provide a minimalistic environment that requires as few arcane buttons, key-presses, and complex controls as possible, yet still offers a great deal of expressive power by drawing on physical metaphors and users' familiarity with the real world. With the *dAb*, *Stokes*, and *IMPaSTo* painting systems, most paintings can be created with just the input device (haptic stylus or tablet) and the space bar on the keyboard. In comparison to the existing computer painting programs, this approach offers the following advantages:

- Natural and expressive mechanisms for manipulating the painting tools, including brushes, palette, paint and canvas;
- Simple and easy loading of complex blends using 3D virtual brushes and the virtual mixing palette;
- Physically-based and realistic brush footprints generated automatically by the brush strokes;
- Intuitive and familiar feel of the painting process requiring little or no training;
- Haptic feedback to give a more realistic feel of painting with a soft brush and to improve the painter's control of brush.

This section presents an analysis of the criteria I have used to design the interface and select the input devices used to control the brush.

8.1 Evaluation Criteria

The best interface for any system is often “the one you already know”. All interfaces require some degree of training to achieve proficiency, but if you already know the interface, or a large part of the interface, then that part of it will be easy to learn and use immediately. In other words, in user interface design, it is worthwhile to leverage existing user training and knowledge when possible.

This is one reason Microsoft and Apple both issue human interface guidelines (HIG) for their platforms. If all developers follow the guidelines, different applications will share behavior wherever feasible. In this way, time spent training to use any one of these HIG-compliant applications serves to increase proficiency in the others. Leveraging training is also why shortcut keys for one dominant application in a market segment often show up in other applications. The makers of competing products know many of their customers will migrating from competitors’ products, so if the interface can conforms to the one users already know, so much the better.

In addition to familiarity, another criterion for a good interface is that it offer a path for a user to develop into an expert, or “virtuoso”. Like a violin virtuoso, a virtuoso with a particular computer interface can get the most out of that application. An interface that is merely easy to learn, but slow and inefficient for novice and expert alike, is not as desirable as one that enables the user to continually improve his or her efficiency, towards the goal of virtuosity. The Microsoft Paint program that comes bundled with Windows is easy for both novices and experts alike, but it is also equally tedious to use for either. In the real world, many activities have the property that their fundamentals are easy to learn, but offer a lifetime of room for improvement. Some examples are dance, chess, martial arts, piano, writing and, naturally, painting.

In the case of painting, for example, the tools are intuitive enough that a child can figure out how to “operate” them to create a painting of some sort, but master painters spend many years perfecting their brush technique, and continue to improve throughout their lives.

8.2 Target User

In designing a user interface, it is also important to understand who the target user is. To take an example from the world of text editors, the Emacs editor is a wonderful interface for a user who is also a programmer, but a very poor interface for a less technical user. Similarly, a simple text editor like Notepad is better for a non-technical user, but not sufficient for most programmers. So the design of an appropriate interface depends to a large degree on the needs of the person who will be using it.

The painting interface I have created is aimed at users who do not traditionally use computer-based painting tools. I have designed the interface specifically not to discourage adoption by novice computer users. The target audience encompasses both those who are interested in beginning to paint, as well as those who already skilled in traditional painting techniques.

8.3 Prior Work on Painting Interfaces

There have been several innovative research and commercial projects that are of interest in the area of interfaces for computer painting. Hanrahan et al. allowed the user to paint directly onto a 3D model by using standard graphics hardware to map the brush from screen space onto the model (Hanrahan & Haeberli, 1990). Commercial systems, such as Z-Brush (Z-Brush, 2000) and Deep Paint (Deep Paint, 2000), also allow users to paint directly on surfaces, by projecting standard 2D brush footprints onto the surface of the 3D objects, similar to Hanrahan and Haeberli's method. The brush itself is not three-dimensional. The idea of painting on 3D surfaces has also been explored in

(Agrawala et al., 1995; Johnson et al., 1999; Gregory et al., 2000a) using a simple, rigid 3D brush controlled by a 6-DOF input device to color 3D surfaces. Most of these 3D painting systems allowed just a simple monochrome brush. Agrawala's interface had the user paint on a physical copy of the virtual object, while the results were displayed on a monitor to the side. (Bandyopadhyay et al., 2001) presented a version of this idea in which projectors were used to display the virtual color directly on the surface of the physical object (which should be painted white). (Adams et al., 2004) enhanced the system in (Baxter et al., 2001) to allow painting on 3D point-sampled models using 3D, deformable brushes.

Several of the more advanced commercial tools, e.g. Painter (Painter 8, 2003), have interfaces that support pen-based input with 5-DOF tablet devices (X,Y,pressure,X-tilt,Y-tilt), yet most tablet-aware applications still only use the position and pressure parameters and ignore the tilt. Further discussion on tablet systems is given in Section 8.5.

8.3.1 The Palette

One of the most effective elements of the original dAb user interface (Baxter et al., 2001) has proved to be the virtual mixing palette. As described in Chapter 4, this is a special canvas dedicated to mixing paint and loading the brush. This general idea is not new to digital painting programs, which is not surprising given the importance of the mixing palette in real painting. In 1985 Quantel patented the idea of a mixing area for digital paint (Searby & Walker, 1985). Unfortunately, Quantel's patent on the idea has likely prevented this interface from becoming widespread in current painting programs. Recently the patent has expired, however, and soon after Corel introduced their own mixing palette interface in Painter (Painter 8, 2003). According to one engineer within Corel who I spoke to, the palette had been available internally for some time, but the

Quantel patent prevented its public release. According to the same source, feedback from users of the new mixing interface has been very positive. One unique feature of the Painter mixing area, as one can determine from simple experimentation, is that it operates with a subtractive color model, probably based on the Zimmer logarithmic dye model patent (Zimmer, 1994). The rest of Painter appears to work with color additively. (Cockshott, 1991) also included a dedicated color mixing area in his user interface. Cockshott mentions that many previous digital paint systems had mixing areas, in addition to Quantel's PaintBox.

The difference between all these previous digital mixing palettes and the ones offered in dAb and IMPaSTo is that the previous ones only allowed the artists to pick a single color from the mixtures created. This completely eliminates one of the primary functions of the real-world palette, which is loading brushes with a complex blend of paints. This is a significant added feature in my palette, because multiple loading is one of the most flexible techniques in a painter's bag of tricks. The technique necessitates a complex brush model, however, which has been lacking in the previous systems that provided a mixing palette.

8.4 Painting Interface

By faithfully simulating the behavior of real paint and real brushes as described in previous chapters, I am able to create a uniquely natural computer interface for painting, as shown in Figure 8.1.

A typical computer painting program presents users with many options or states in order achieve a wide variety of effects. This type of interface is to some degree necessitated by the limited input modality and limited expressiveness of the digital media these programs provide the user.

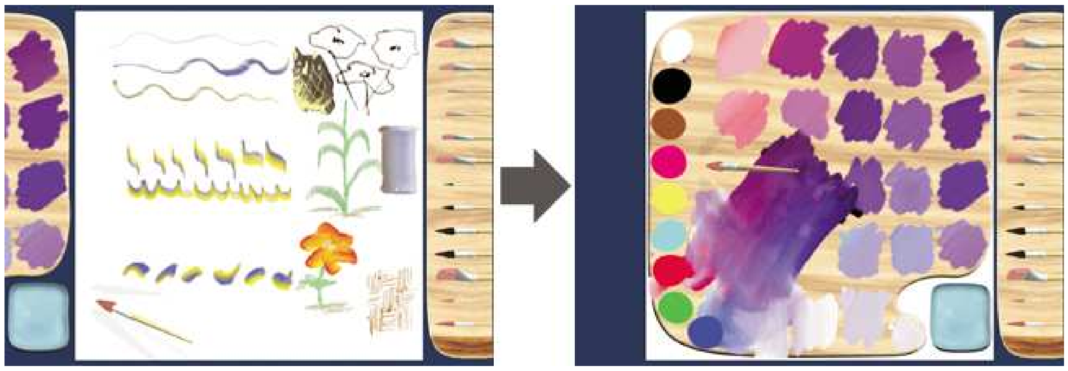


Figure 8.1: *Graphical User Interface: The virtual canvas with the brush rack and a portion of the palette (LEFT); the brush rack and the palette for color mixing (RIGHT).*

For instance, in typical painting programs, in order to create a paint stroke that transitions between two colors it is often necessary to open a special dialog that controls color gradients, and select the two colors that will be part of the stroke, and then define the direction of the color transition. In contrast to that procedure, a traditional painter with traditional tools would simply load a little bit of both colors onto his or her brush and begin to paint the stroke directly.

With my interface the painter is able to duplicate the traditional workflow, because the brush loading, paint motion, and paint blending are all simulated realistically. Thus the user is able to load the brush just as he would a real brush.

This natural interface is the perfect choice given the desire to make the interface accessible to non-technical users. The interface presents the user with nothing but the basic tools themselves: brushes, canvas, paint, and palette. The rest of this section gives an overview of the key aspects of the natural interface.

8.4.1 The Brushes

The brush modeling was discussed in Chapter 3. The ability to control a 6DOF, 3D virtual brush in the painting simulation is critical to the natural interface, and is one of the key differences between the interface I present and that of typical computer painting programs.

The input device for controlling the brush is an important factor in giving the interface a natural feel. I have experimented with several different input devices as described later in this chapter.

8.4.2 The Paint

Chapters 4–6 presented three different models for paint. As Tia from Pixar complained in the introduction, with typical paint programs, “you can’t push junk around.” The paint models I have presented, in particular Stokes and IMPaSTo, truly allow the painter to work with the paint media as if it were a real physical substance, allowing for a very natural means of achieving a desired distribution of paint on the canvas.

8.4.3 The Canvas

The canvas is where the user’s focus is primarily directed most of the time. Thus I give the canvas the majority of the screen real estate. The normal view presented to the user is shown on the left of Figure 8.1, with a small brush rack on the right and, and a portion of the palette visible (and usable) on the left.

With a tap of the space bar, or by waving the brush off the screen in the direction of the palette, the palette slides out for mixing as shown on the right of Figure 8.1.

8.4.4 The Palette

Despite the palette being an intuitive interface for choosing and creating colors, this paradigm has not been widely supported in most previous computer painting systems.

One of the main advantages of my 3D brush and palette combination, however—and one not possible with any previous system—is that loading the brush with complex blends is simple. To get a blend of colors one simply needs to drag different parts of the brush through different colors to pick them up, and then the stroke created on the canvas will consist of those colors blended together as expected. With a conventional program, if such an operation is supported at all, it requires defining a gradient fill for the stroke, and it still will not offer the same level of intuitive control over the blend effect afforded by simply rotating the 3D brush during a stroke, for instance.

8.4.5 Brush Shadows

One feature initially missing from the interface, but that was quickly found to be indispensable, was brush shadows. Since the virtual brush is a 3D object in the scene, it can be difficult to determine from a single-perspective view on the monitor exactly how far the brush is away from the canvas. The simple solution was to provide depth cues in the form of shadows. In fact, there should be several shadows, to increase the chances that at least one of them will be on-screen and be casting onto a light enough region of the canvas where it has best visibility. In the end I also added an "inverse shadow" capability, which lightens rather than darkens, after one user decided to paint a completely black picture of outer space.

8.4.6 Bi-manual Input

Several studies have shown improved task performance when using interfaces that allow the use of both hands (Owen et al., 1998; Buxton & Myers, 1986; Kabbash et al., 1994; Kabbash et al., 1993). Specifically, what works well is when the non-dominant hand is used for secondary tasks like positioning a see-through overlay tool (Bier et al., 1994), or for positioning the surface on which the user is writing.

Since we use an input device other than the mouse as the primary input, as will be described below, this leaves the mouse free for use in the user's non-dominant hand. In my implementation of the interface, I allow the user to use the mouse to select brushes, operate menus, and to position the palette and canvas. In this way the user can wield the brush with the dominant hand and at the same time place the canvas or position the palette with the non-dominant hand, exactly the sort of task at which it excels.

8.5 Input Devices

In order to allow painting to be performed seamlessly one needs an input device capable of expressing the motion of a 3D virtual brush. There are a few technologies on the market today that I considered for this purpose. My interface supports two of them: the Phantom haptic interface from SensAble Technologies, and the Wacom Intuos Tablet interface.

There are three main characteristics desired from a brush control input device. First, the number of input degrees of freedom (DOF) should be as close to the six degrees of freedom of an actual brush as possible. Second it should be approximately the same bulk as a brush to allow for the same sorts of control and range of motion that is possible with a brush. Finally, the device would ideally be able to generate haptic feedback similar to an actual brush, either by virtue of being an actual brush, or via programmatic force feedback. We summarize some of the input devices available in Table 8.1.

We will first describe the pros and cons of several of the most common types of input device.

8.5.1 Mouse

The mouse is the input device used most commonly in digital painting applications today; however, it is far from being the ideal input device for painting. The limited

IO Device	Input DOF	Output DOF	Bulk	Haptics
Mouse	2	0	Low	None
6DOF Phantom	6	6	High	Programmatic
3D Tracker	6	0	Medium	None
Wacom Intuos Tablet	5	0	Low	Static
Desktop Phantom	6	3	Medium	Programmatic
Omni Phantom	6	3	Medium	Programmatic

Table 8.1: *Various input devices that can be used for brush control.*

degrees of freedom of the mouse mean that applications must provide other means to control the appearance of strokes, which almost always are less intuitive than using a real brush.

8.5.2 Trackers

3D trackers employ several different technologies and are available in many configurations. Major technology categories are optical, magnetic, electrical and inertial. There are three main drawbacks with existing tracker technologies. The first is bulk. Existing tracking devices are often too bulky to place on a paintbrush (as with the UNC Hi-Ball optical tracker), and nearly all require a tether to a power-supply, which is undesirable. The second drawback is that they provide no haptic response. They are completely free-floating in space, though static haptics can be constructed. The third drawback is that most trackers are designed for tracking a large area, and they do this at the expense of fine scale accuracy and response time. The working volume of a typical painting does not match the working volume of most trackers well.

8.5.3 Tablets

Several manufacturers make tablets for pen-based input. There are a wide variety of these, with some offering only the same number of input degrees of freedom as a mouse (X-Y position only). Many offer a third degree of freedom in the form of a pressure value, which can be mapped to a Z coordinate. The Wacom Intuos and Intuos2 lines of tablets are capable of measuring an additional two degrees of freedom: X and Y tilt angles, for a total of five DOF input.

This is still one DOF fewer than is necessary to control a 6DOF brush (it lacks a twist DOF), but it is a reasonable interface for painting despite the missing degree of freedom. One feature of the tablet is the pen-like feel. On the positive side, the haptic response is real, so there is no latency and the friction is real, though somewhat plastic. On the negative side, however, it always feels like a pen on hard surface, and it is not possible to programmatically make the brush feel stiffer or softer, or to feel the virtual paint.

From a convenience point of view, some users find this interface easier to control because of the real friction, and because it gives them a place to rest their hand, and through that resting point, a better proprioceptive sense of context.

8.5.4 Haptic Devices

While many haptic devices have been designed and built (see (Hayward et al., 2004) for a recent survey), the Phantom (Salisbury et al., 1995a) is perhaps the most commonly used, and it provides a very good platform for development. The Phantom family of devices are all 6-DOF input, based on an articulated armature design. The Desktop and Omni models provide 3-DOF force feedback, while the Premium-A model is capable of full 6-DOF haptic feedback (force and torque).

The 3-DOF Desktop and Omni models are the best suited to use as painting interfaces since their armatures are the least bulky, and have inertial properties more similar to a paint brush than the larger models. Still, the bulk and the armature that tethers them to the base are drawbacks. Another drawback is the lack of hand rest, which some users find to be tiring. On the other hand, there is typically no hand rest in actual painting, either.

The unique feature of the haptic devices is that they are able to deliver an arbitrary force to the user under programmatic control. Thus the brush properties can be modeled, and the feel of different brushes and paint can be simulated. This is the focus of the next section.

Part II: Haptic Feedback

"Where the spirit does not work with the hand there is no art."

— Leonardo da Vinci

Brushes have thousands of individual hairs, each of which can move independently. Too much pressure applied in the wrong direction can cause a brush to splay in an undesirable way, or, alternatively, that may be exactly the effect the painter wishes to achieve. Either way the painter needs feedback from the brush in order to control it properly. Visual feedback is certainly one important channel of information, which is why in my interface I display the entire 3D virtual brush, rather than just the active portion that is currently depositing paint. But haptic sensations are also a very important type of feedback. The direction and amount of force felt by the painter gives a very useful indication of the brush's state at every instant. A skilled painter could probably visualize what a stroke will look like without looking based on the feel alone. Thus good haptic feedback is a useful addition to a painting system.

Haptic feedback using the Phantom interface is one of the main novel aspects of my painting simulations. I attempt to provide sufficiently good force feedback to emulate the sensation of applying brush strokes to a canvas. The 6-DOF armature input device also serves as a 3-DOF force output device.

I have investigated two methods for delivering haptics, the first, used in dAb, was a relatively simple, but still quite effective mechanism, and the second technique is based on the more physically accurate computation possible when simulating fluid-based paint with Navier-Stokes or Stokes equations.

For all force computations, I align the virtual paintbrush with the physical 6-DOF stylus, and position it so that the point of 3-DOF force delivery coincides with the point where the brush head meets the handle on the virtual brush.

8.6 Related Work on Haptics

In this section, I present a brief survey previous work related to haptic rendering, applications of haptics, and force computation in simulated fluid-structure interaction.

8.6.1 Force Feedback

Several techniques have been proposed for integrating force feedback with real-time virtual environments to enhance the user's ability to perform interaction tasks (Colgate & Brown, 1994; Massie & Salisbury, 1994; Salisbury et al., 1995b).

(Ruspini et al., 1997) presented a haptic interface library "HL" that uses a virtual proxy and a multi-level control system to effectively display forces using 3-DOF haptic devices. Hollerbach et al. (Hollerbach et al., 1997; Nahvi et al., 1998) described a haptic display system for contact and manipulation in the CAD design of mechanical assemblies and Thompson et al. (Thompson et al., 1997) have presented a system for direct haptic rendering of sculptured models.

Techniques for haptic visualization of the topology of vector fields were investigated by Helman and Hesselink (Helman & Hesselink, 1990; Helman & Hesslink, 1991). Durbeck et al. (Durbeck et al., 1998) have described a system for enhancing scientific visualization by the use of haptic feedback. The combined haptics/graphics display is used for displaying flow fields, and vector fields. These systems were based on 3-DOF haptic devices that provided only force feedback. Recently, Lawrence et al. presented a technique for shock and vortex visualization using a combined visual and haptic interface with a 5-DOF force feedback device (Lawrence et al., 2000).

In (Yeh et al., 2002) the authors propose an evaluation of haptics for painting and present the results of a preliminary study comparing two conditions: with and without haptics. The haptics are a modification of the model presented in (Baxter et al., 2001). The study consisted of a simple yes/no questionnaire asking each of the participants which condition was better. The results are fairly inconclusive since there were only six participants. However, four out of the six subjects preferred the presence of haptics. The results of any such study are obviously highly dependent on the quality of the actual haptic implementation used. They report that a more extensive study is planned.

8.6.2 Volumetric Approaches

Gibson (Gibson, 1995) proposed an algorithm for object manipulation including haptic interaction with volumetric objects and physically-realistic modeling of object interactions. The algorithms presented by Avila and Sobierajski (Avila & Sobierajski, 1996) rely on interactive force feedback and rendering to allow a user to quickly explore and modify volumetric scenes.

Essentially, most present techniques for direct haptic rendering of volumetric data follow a general strategy in which the force display is defined as a vector-valued function, or is transformed to one (Avila & Sobierajski, 1996; Iwata & Noma, 1993; Gibson, 1995). From this vector function, force feedback is generated from the data around the probe and from the velocity of the tip.

8.6.3 6-DOF Haptic Rendering

Iwata describes a 6-DOF haptic master and the concept of time-critical rendering at a lower update rate of hundreds of Hz (Iwata, 1990). Iwata and Noma also proposed methods for presenting volume data by force sensation using a 6-DOF force reflecting master manipulator with an update rate of a hundred Hz (Iwata & Noma, 1993).

Recently (McNeely et al., 1999) proposed "point-voxel sampling," a discretized approximation technique for contact queries that generates points on moving objects and voxels on static geometry. This approximation algorithm is the first to offer run-time performance independent of the environment's input size by basically sampling the object geometry at a resolution that the given processor can handle.

A recent approach proposed in (Gregory et al., 2000b) is limited to haptic display of object-object interaction for relatively simple models that can be easily represented as unions of convex pieces. (Kim et al., 2002) attempts to increase the stability of the force feedback using contact clustering, but their algorithm for contact queries suffers from the same computational complexity. (Otaduy & Lin, 2003) introduces a "*sensation preserving*" simplification algorithm for faster collision queries between two polyhedral objects in haptic rendering, thus achieving time-critical 6-DOF haptic rendering for highly complex models and contact configurations.

8.6.4 Fluid Force Computation

Foster and Metaxas (Foster & Metaxas, 1996) used a form of the hydrostatic force equations to create animations of rigid body objects in their offline Navier-Stokes simulation. Hydrostatics ignore the dynamic effects of fluid flow on the objects. Their simulation also did not account for the effect of the objects on the fluid.

Tu computed forces by using a boundary integral of the relative velocity between a fish's fin surface and surrounding fluid for simulating how swimming motions propel a fish (Tu, 1996). Like (Foster & Metaxas, 1996) this method apparently does not take into account the influence of the immersed surface on the fluid, since at the actual surface of an immersed object, the physical boundary conditions demand that the velocity of the fluid relative to the surface is always zero, which would mean the force should always be zero as well with this method.

Ristow (Ristow, 1999; Ristow, 2000) has created several offline simulations of spherical and elliptical particles falling through fluids using accurate force computations based on the fluid stress tensor. The method I propose is based on the same equations but uses a different numerical procedure to compute the force. A further distinction is that in my case the "particle" is an actively controlled haptic probe rather than a passively simulated object.

To the best of my knowledge, no one to date has used real-time fluid simulation to generate the force feedback to drive a haptic display, allowing the user to both feel and influence the fluid at the same time.

8.7 Simple Force Computation

8.7.1 Decoupled Haptics

In the simple force model, I separate the force computation from the brush deformation computation, since the two have different goals. For instance, the force updates for haptic feedback need to be generated at close to 1kHz for smooth jitter-free output, but the deformation calculation only needs to proceed at visual update rates (around 30Hz). Consequently, in the simple force model I solve this problem by decoupling the force simulation from brush dynamics simulation, and simplify the force computation to run at kHz rates.

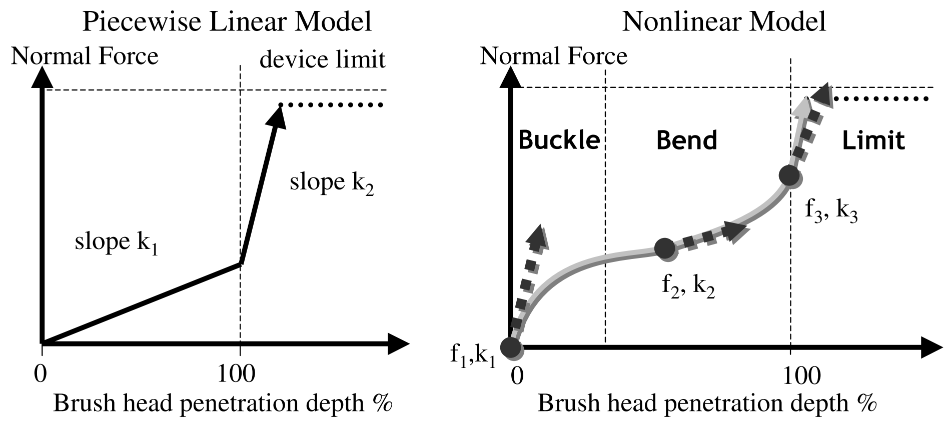


Figure 8.2: *Example graphs of the normal force used in the basic force model and non-linear force model.*

8.7.2 Basic Force Model

The root of the force model is a simple piecewise linear function of the penetration depth of the undeformed brush point. If d_p is the penetration depth , and l_p is the length of the brush head projected onto the canvas normal, $\mathbf{n}$, then the force is modeled as:

$$\mathbf{f}_b(d_p) = \begin{cases} 0 & \text{if } d_p \leq 0 \\ \mathbf{n}(k_1/l_p)d_p & \text{if } 0 < d_p \leq l_p \\ \mathbf{n}(k_1 + (k_2/l_p)(d_p - l_p)) & \text{if } l_p < d_p \end{cases} \tag{8.1}$$

where k_1 is a small positive constant that models the light spring of bristles and k_2 is a larger positive constant that simulates collision of the actual brush handle with the canvas. The spring constants are normalized by l_p so that the same absolute force is delivered when the handle first hits the canvas, regardless of the brush length or orientation. The value of k_1 can be changed to simulate brushes of varying stiffness. See Figure 8.2.

Compressive Effects

When a real brush contacts the canvas at close to a right angle, the stiff bristles initially act as strong compressive springs, transmitting an abrupt force to the handle. As more pressure is applied, the bristles buckle and the compressive force reduces as bending forces take over. When the brush makes a contact at an oblique angle, compressive effects play a lesser role in the force felt.

Therefore, I extend the piecewise linear function, Equation 8.1, to a piecewise Hermite curve, as shown on the right of Figure 8.2. This curve is defined by a series of control tuples that contain the penetration depth and corresponding force magnitude, and the linear stiffness of the spring model at that point. I currently use a four-segment piecewise curve, which was derived from the empirical observation of how a brush head behaves under compression.

The initial segment of the piecewise curve models the compressive force. I assign the initial control tuple a fairly strong linear spring constant to simulate the initial strong compressive force. I modulate this compressive force according to the angle of contact, by multiplying the force value of the second control tuple by an angle-dependent coefficient between one and zero. Given θ, the angle between the canvas normal and negated bristle direction vector, the factor I use is

$$\gamma = \begin{cases} \cos^2(2\theta) & \text{if } -\frac{\pi}{4} < \theta < \frac{\pi}{4} \\ 0 & \text{otherwise} \end{cases} \tag{8.2}$$

This results in a compressive force that is strongest when a brush contacts the canvas at a right angle and that tapers off to zero as the brush approaches a 45 degree angle to the canvas.

Frictional Forces

An important component of the force delivered to the user is a small amount of tangential resistance. Though small in magnitude, frictional forces have a large effect on the user's perceived ability to control the brush by damping small oscillations in the user's hand. I model friction $\mathbf{f}_t$ simply, as a force opposite the current brush velocity, $\mathbf{v}_b$, which is added to the other feedback forces:

$$\mathbf{f}_t = k_t \left(\mathbf{v}_b - \mathbf{n}(\mathbf{n} \cdot \mathbf{v}_b)\right)$$

where k_t is the coefficient of friction.

8.8 Fluid Force

In this section I will describe a method of computing haptic feedback from fluid simulations I have developed that is applied to the fluid-based painting simulation, but is actually a general method applicable to any fluid simulation that generates velocity and pressure fields as output.

8.8.1 Preliminaries

The motion of an incompressible Newtonian fluid is described by the Navier-Stokes equations. The momentum equation describes the transport of momentum in the fluid:

$$\frac{\partial \mathbf{u}}{\partial t} = -(\mathbf{u} \cdot \nabla)\mathbf{u} - \frac{\nabla p}{\rho} + \nu \nabla^2 \mathbf{u} + \mathbf{F} \tag{8.3}$$

where $\mathbf{u}$ is the fluid velocity field, p is the pressure field, ρ is the density, ν is the coefficient of kinematic viscosity, and $\mathbf{F}$ is an external force per unit mass such as supplied by gravity. The dynamic viscosity will also be needed later, which is $\mu =$

$\rho\nu$. Incompressibility adds an additional constraint, known as the continuity equation, which for a constant-density fluid can be written:

$$\nabla \cdot \mathbf{u} = 0 \tag{8.4}$$

Appropriate boundary conditions are necessary as well. For a "no-slip" boundary condition on velocity and pure Neumann boundary condition on pressure, in a domain Ω with boundary $\partial\Omega$, these are

$$\mathbf{u} = 0 \in \partial\Omega \tag{8.5}$$

$$\frac{\partial p}{\partial \mathbf{n}} = 0 \in \partial\Omega \tag{8.6}$$

In this section I present methods for computing a force for haptic feedback from a fluid simulation. Before presenting the accurate method, I first look at some common force approximations used in fluid dynamics.

8.8.2 Approximated Fluid Forces

For rough aerodynamic calculations of lift and drag forces, a few simple approximations are commonly used. One standard equation used to approximate the aerodynamic drag force on an object with relative airspeed v (Bloomer, 2000) is

$$F_{\mathrm{AD}} = C_{\mathrm{AD}}\ \rho v^2 A, \tag{8.7}$$

where A is the projected cross-sectional area of the object, C_{AD} is an aerodynamic drag coefficient that depends upon the shape and material properties of the object, and ρ is the fluid density. this approximation is only applicable to low-viscosity, and low vorticity situations such as the laminar air stream in a wind tunnel.

For high-viscosity, or slow flow scenarios in which the Stokes approximation holds, one can obtain a similar expression for the viscous drag force(Landau & Lifschitz, 1975):

$$F_{\mathrm{VD}} = C_{\mathrm{VD}}\ \mu v A \tag{8.8}$$

where C_{VD} is a coefficient of viscous drag that depends on the shape of the object, and μ is the viscosity of the fluid. This type of simple linear relation between force and velocity has been used quite extensively in haptics to simulate viscous or dynamic friction, because it is simple, computationally inexpensive, and quite stable. In such uses, the source of the viscous force is generally taken to be at rest so that the probe's velocity is the overall relative velocity.

Both of these approximations make the assumption that the velocity field can be characterized by a single vector, so these methods can only generate forces, not torques. If a single sample velocity from the flow field is chosen, the method becomes similar to that used by many haptic volume display systems, where a vector field is haptically rendered using a force proportional to the vector at the probe location.

One can use these equations for approximating the fluid force on a point probe, but difficulty arises if you wish to also interact with the fluid via the same point probe. If interaction is implemented by simply injecting velocity into the simulation at the probe location, as is common, then this injected velocity becomes a large part of what gets sampled and the resulting force is incorrect. A better, more physically correct approach is to cause the fluid to move via boundary conditions on a finite probe. This will be discussed more in Section 8.8.4.

8.8.3 Accurate Force Computation

The proper description of the internal forces in a viscous incompressible ideal fluid is given by the stress tensor, $\boldsymbol{\sigma}$, which in index notation is given by (Landau & Lifschitz, 1975)

$$\sigma_{ik} = -p\delta_{ik} + \mu\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}\right). \tag{8.9}$$

The force per unit area at a given point in the fluid, $\mathbf{x}$, on an infinitesimal area dA with normal $\mathbf{n}$ is given by:

$$\mathbf{P} = \boldsymbol{\sigma}(\mathbf{x}) \cdot \mathbf{n}; \tag{8.10}$$

The net force acting on a closed object submerged in the fluid can then be obtained by integrating this expression over the surface of the object:

$$\mathbf{F}_{\text{obj}} = \int_S \boldsymbol{\sigma} \cdot \mathbf{n}\, dA. \tag{8.11}$$

In terms of vector components, $\mathbf{P}$ is given by

$$P_i = (\boldsymbol{\sigma} \cdot \mathbf{n})_i = -pn_i + \mu\left(\frac{\partial u_i}{\partial x_k} + \frac{\partial u_k}{\partial x_i}\right) n_k \tag{8.12}$$

or, for a two dimensional viscous flow,

$$\begin{pmatrix} P_x \\ P_y \end{pmatrix} = \begin{pmatrix} -p_x n_x + \mu\left(2\frac{\partial u_x}{\partial x} n_x + (\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}) n_y\right) \\ -p_y n_y + \mu\left(2\frac{\partial u_y}{\partial y} n_y + (\frac{\partial u_x}{\partial y} + \frac{\partial u_y}{\partial x}) n_x\right) \end{pmatrix} \tag{8.13}$$

Similarly, the torque on the object about the point $\mathbf{c}$ is given by integrating the cross product

$$\boldsymbol{\tau}_{\text{obj}} = \int_S \mathbf{r} \times \boldsymbol{\sigma} \cdot \mathbf{n}\, dA. \tag{8.14}$$

where $\mathbf{r}$ is the vector from $\mathbf{c}$ to each infinitesimal surface element. For computing the haptic force, $\mathbf{c}$ acts as the point of attachment of the haptic stylus to whatever probe geometry is desired. In two dimensions where torque is a scalar, one obtains:

$$\tau_{\text{obj}} = \int_S (r_x P_y - r_y P_x)\, dA. \tag{8.15}$$

Equations 8.11 and 8.14 tell everything one needs to know to generate haptic feedback from a fluid solution. The next step is to determine an appropriate discretization of these equations in order to evaluate them numerically on a computational grid, which will be discussed in the next section.

8.8.4 Haptic Display of Viscous Flows

There are numerous ways to discretize Navier-Stokes equations, with various strengths and weaknesses. For my purposes I desire a simulation method that emphasizes speed and interactivity. For this reason it is important to choose a method without timestep restrictions. Many solution methods require for stability that the simulation timestep be selected such that a CFL condition such as $\Delta t\, \mathbf{u}_{i,j} < \Delta \mathbf{x}$ holds for all (i, j) on the discretized grid. Even worse are timestep restrictions on viscosity that typically require $\Delta t < O(\Delta x^2 \nu^{-1} \|\mathbf{u}_{i,j}\|^{-1})$. Even if the basic time-stepping procedure is fast, the number of steps required to advance the simulation by a desired time increment ΔT can be prohibitive, and cause the simulation clock to fall behind the wall clock time.

A better approach for interactive fluid simulation is that presented by Stam et. al. (Stam, 1999). By using an implicit time-stepping procedure to handle viscosity, and a semi-Lagrangian scheme for advection, all stability restrictions on the timestep are removed, and thus it is possible to prevent the simulation clock from falling behind the wall clock.

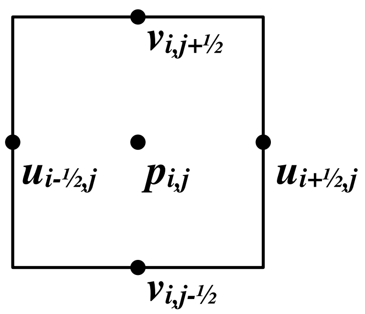


Figure 8.3: *A typical grid cell, at grid location* (i, j).

The approach is a fractional step method with a Chorin projection (Chorin, 1968) to solve for the pressure. The differential operator is approximated by splitting it in time so that the advection term, viscosity term, and pressure terms are handled sequentially instead of simultaneously. The incompressibility constraint is ignored for the first substeps, and then the resulting flow field is projected onto the space of divergence-free vector fields to satisfy the continuity equation. This approach requires the solution of a discretized Poisson problem. The overall procedure is first-order accurate using first-order discretized operators.

I implement the solver for the most part like Stam, except I use the conjugate gradient method with an incomplete Cholesky preconditioner to solve the sparse linear systems of equations that arise from the implicit viscosity step and the Poisson equation for pressure (Kershaw, 1978). Also I use a staggered grid rather than a cell-centered grid, since the Poisson solver tends to converge better on a staggered grid (Griebel et al., 1990). A typical grid cell is shown in Figure 8.3.

To discretize the boundary of the haptic probe, I use the common first-order approach of treating each grid cell as either entirely fluid or entirely solid, thus snapping the actual boundary shape to the nearest grid cell edges as shown in Figure 8.4. This is the same approach as proposed in the original Marker-and-Cell method (Harlow & Welch, 1965).

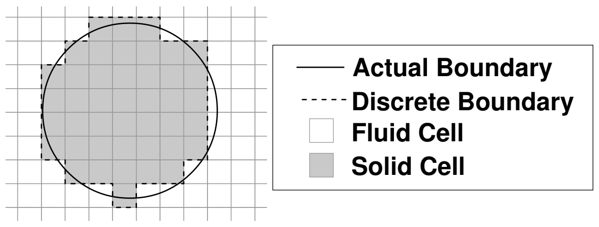


Figure 8.4: *Boundary discretization used for incorporating the haptic probe geometry into the grid-based simulation.*

To incorporate the motion of the haptic probe into the simulation, the u and v velocity boundary values are set along the perimeter of the discretized probe boundary on the grid. For a north facing edge, for example, I enforce the no-slip boundary condition by setting

$$\begin{aligned} v_{i,j+\frac{1}{2}} &= v_{\text{probe}}\left(i\Delta x, (j+\tfrac{1}{2})\Delta y\right) && (8.16)\\ u_{i\pm\frac{1}{2},j} &= 2u_{\text{probe}}\left((i\pm\tfrac{1}{2})\Delta x, j\Delta y\right) - \\ &\quad u_{i\pm\frac{1}{2},j+1}, && (8.17)\end{aligned}$$

and similar expressions are used for the other cell edges.

8.8.5 Force Computation

After an iteration of the numerical fluid solver, I have values for p, and and $\mathbf{u} = (u, v)$ on the grid. To compute the force and torque on the probe, at this point I simply need to traverse the boundary and and approximate the boundary integrals Equations 8.11 and 8.14, using a discrete Riemann sum. For a north facing edge, for instance, the normal pointing in toward the probe is $\mathbf{n} = (0, -1)$ and the contributions to the total force and torque obtained by using this $\mathbf{n}$ in Equations 8.11-8.15 are

$$\begin{aligned}
\begin{pmatrix} P_x \\ P_y \end{pmatrix} &= \begin{pmatrix} \mu\frac{v_{i,j+3/2}-v_{i,j+1/2}}{\Delta x} \\ -p+2\mu\frac{v_{i,j+3/2}-v_{i,j+1/2}}{\Delta y} \end{pmatrix} \qquad (8.18)\\
\mathbf{F}_i &= \Delta x \begin{pmatrix} P_x \\ P_y \end{pmatrix} \qquad (8.19)\\
\tau_i &= \Delta x \left(r_x P_y - r_y P_x\right). \qquad (8.20)
\end{aligned}$$

Similar expressions are obtained for the remaining three edge orientations. After traversing the entire boundary of the probe, one obtains $\mathbf{F}_{\text{obj}} = \sum_i \mathbf{F}_i$ and $\tau_{\text{obj}} = \sum_i \tau_i$.

8.8.6 Force Filtering

The above method does suffice for calculating the force from the flow, but generally in the simulations of interest–a 64×64 grid or larger–the fluid simulation cannot execute at the 1KHz rate desired for smooth haptic rendering on a typical GHz-class processor. 40-70Hz is a more typical simulation rate for such a grid. For general haptics, involving the display of rigid contact with rigid surfaces, any sort of smoothing filters can lead to an undesirable softening of the surfaces by removing the high frequency components of the force displayed. However, for simulating the feel of a fluid, the high frequency components contain much less energy, and I have found some amount filtering to be both acceptable and beneficial in reducing the artificial stair-stepping and vibration that comes from updating the haptic output at less than the desired 1KHz rate.

In selecting a filter, a desirable characteristic is a step response function that smoothly transitions both away from the previous state and into the new state, to avoid introducing high frequency artifacts. Finite impulse response (FIR) window

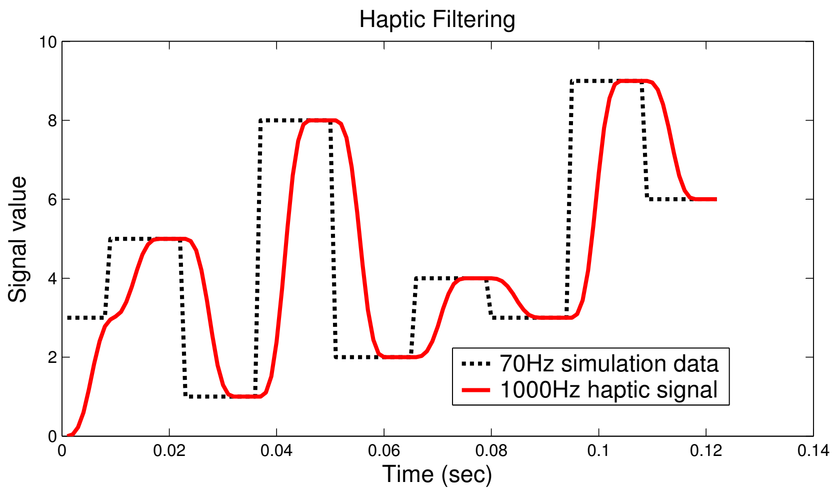


Figure 8.5: *The results of filtering a random 70Hz input signal for haptic display at 1KHz.*

filters match these desired characteristics well. I selected a 10th order Bartlett-Hanning window using the Matlab signal toolbox. My results indicate that filtering the force in this manner yields a notable improvement in the smoothness of the haptic feedback for this application. The filter's smoothing effect on an input signal can be seen in Figure 8.5. Although the filtering does introduce some delay into the force response time, the users did not find this to be noticeable in the case of fluid force display.

8.9 Haptic Results

I have implemented the force feedback methods as described above in C++ and tested this implementation on a 2GHz Pentium IV computer with a Phantom haptic interface from SensAble Technologies, Inc. For force only, I used a Desktop Phantom model with 3DOF of force feedback. For force and torque, I used a Premium-A 6-DOF model Phantom.

8.9.1 Simple Haptics

The simple haptic feedback model—either piecewise linear or non-linear—can easily be calculated at the desired rate of 1KHz on the test system above. Though the model is simple, the simplicity of the computation makes for a very smooth and consistent springy brush response. The non-linear model makes the brush feel less compliant on initial contact more like a real brush.

8.9.2 Fluid-Based Haptics

For fluid based haptic feedback, in the implementation I control the final scaling of the fluid force with a user-defined parameter. This parameter should be chosen carefully since too low a value will make the fluid forces too weak, but too high a value can add excessive energy back to the haptic probe that is then fed back to the fluid leading to unstable oscillations. The scale used in my simulation is typically in the range of 2-10. This is for a unit domain of $[0, 1] \times [0, 1]$ at 64×64 resolution with a Reynolds number of about 10^3.

Figure 8.6 shows a few frames from a fluid interaction sequence in which a square probe is swept along a path. In Figure 8.7 I show the force as computed by my simulation method along that path. The square begins at rest, accelerates as it sweeps out an arc, then comes to rest. Then it reverses direction and traces the same path backwards. At the same time the square spins with an angular velocity that has a similar profile of acceleration. I have split the 2D force into two components, one the drag force, tangential to the motion of the probe, and the other component is perpendicular to the motion.

The force and torque profiles can be seen to contain some amount of noise. This jitter comes from the discretization of the boundary. Figure 8.8 shows how the noise is reduced when I sweep along the same path repeatedly at different grid resolutions.

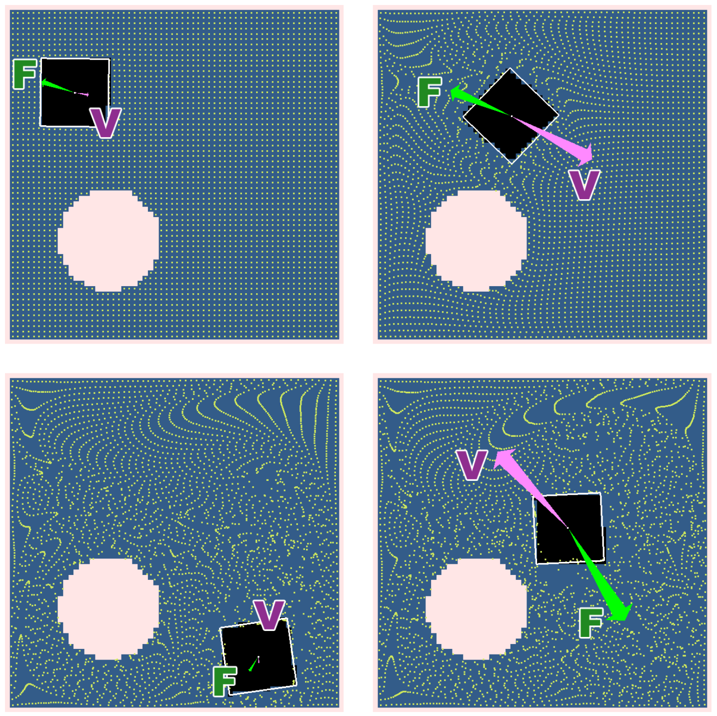


Figure 8.6: *A sequence of images captured from my interactive fluid simulation with haptic rendering, showing both the instantaneous force (green arrows) and velocity vectors (magenta arrows) at several instants along the probe's path.*

Although the simulation does not run interactively at the higher resolutions, the reduction in noise indicates that boundary discretization is the major source of the noise. Higher order numerical treatments of the boundary have been developed, such as (Tome & McKee, 1994), and these would likely reduce the noise in the force signal. In my current implementation this jitter is not very noticeable since the magnitude of the variation is only about 0.005N.

To demonstrate the practical utility of my method in virtual applications I have also integrated the force computation into my painting system using the Stokes paint model from Chapter 5. Using this method, a user is able to feel the viscosity of the the paint as he or she makes brush strokes using the haptic stylus. See Figure 8.9.

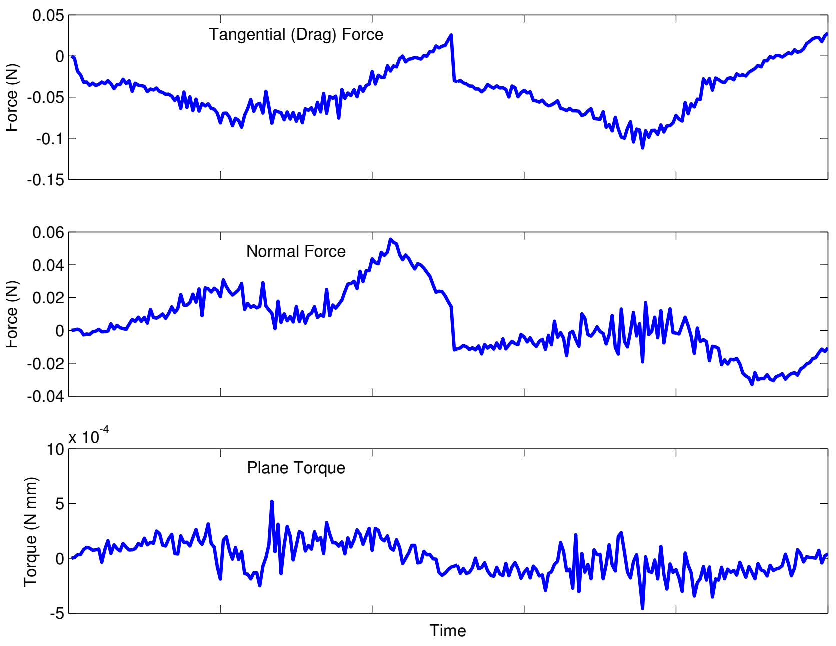


Figure 8.7: *Unfiltered haptic feedback forces generated by my method for a square probe dragged and twisted along a path. My haptic filtering method smooths these values for the final force display. The object first accelerates from rest, decelerates to a stop at the middle of the graph, and then returns to its starting point (see Figure 8.6). The drag force graph (top) clearly shows the fluid force working against the direction of motion, as expected.*

8.10 Limitations and Future Work

The ideal haptic feedback for a paint simulation system should be indistinguishable from the feel of a real brush, and every nuance of the elastic force of the bristles, every aspect of the friction of the bristles sliding against a rough canvas, and the every change in force due to interaction with paint medium should be modeled. There is still much ground to cover before reaching this goal, but the methods for brush haptic feedback simulation presented in this dissertation present several strides in that direction, and serve as a proof of concept for the value of haptic feedback in this type of application. My implementation of the concept is compelling enough in its current form that SensAble Technologies, the makers of the Phantom haptic device, have even

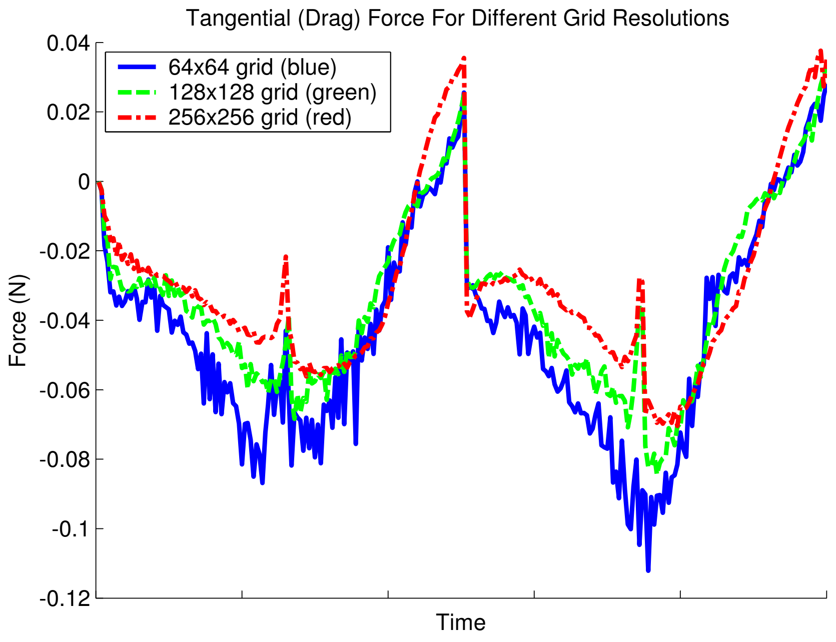


Figure 8.8: *The effect of grid resolution on the force computed. The noise in the signal is reduced with each increase in grid resolution from the coarsest grid (blue) to the finest grid (red).*

begun to use my painting systems as a demonstration of their haptic devices at trade shows.

My demo system has the ability to switch between different test haptic feedback modes for comparison. For instance, there's a mode that disables the haptic force entirely, and another that emulates the type rigid contact felt when using a tablet interface. Practically every single user I have asked has expressed a preference for my standard springy haptic model compared to no haptics or rigid haptics. These results differ from the preliminary results presented by (Yeh et al., 2002), who found only 2/3 of users in a small study to prefer painting with haptics. A formal study, and perhaps further research into realistic haptic models, will be required to truly determine the value of haptics in applications such as this.

Concerning the interface as a whole, the prototype system I have put together is just a demonstration of the potential for this type of natural, realistic interface for a

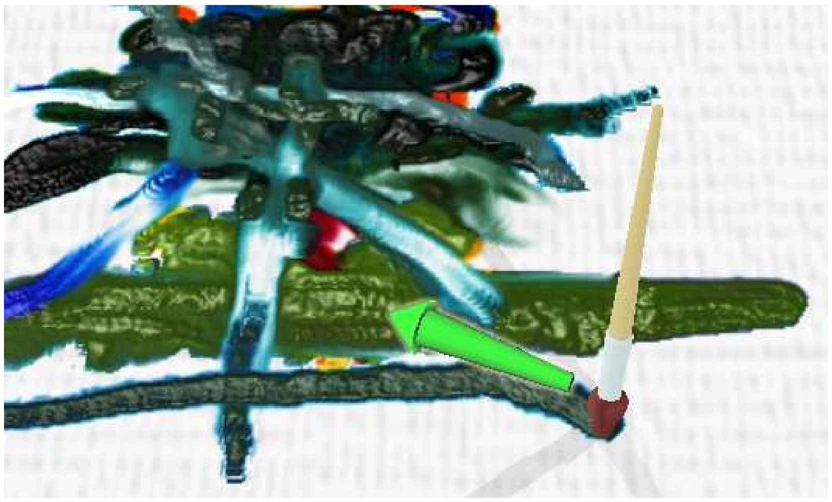


Figure 8.9: *This is a screen captured from a painting application modified to use my method to display the feel of the viscous paint to the user. The green arrow shows the force being displayed as the user paints a stroke from left to right.*

painting system. It currently lacks many of the features that may be required by serious professional users in the long term, such as flexible support for layering and masks, and a variety of import and export options. These are probably features that need to be added; however, the current prototype system is still very capable of creating works of art, as can be seen in the many example paintings shown in the previous chapters.

Based on the feedback from the hundreds of people who have tried my painting systems, overall it is clear that the interface has accomplished the goal of appealing to users who are interested in painting, but not necessarily computer painting, and also of being easy to learn and use. Children, in particular, seem to really enjoy the system and are able to work with it immediately, with very little instruction, though professionals such as John Holloway (see Appendix B) have also found the interface rewarding to work with. Non-technical artists also seem to be very enthusiastic about it.

8.11 Summary

This chapter has presented the natural painting interface I have developed using the physically-based models presented in the previous chapters. I have also presented two models for haptic feedback: a simpler, spring-based model decoupled from the actual brush and fluid simulations, and a method based on the stress tensor for computing force directly from fluid simulation data.

In summary, in this chapter I have presented:

- A natural interface for a computer painting system that combines a mixing palette and 3D deformable brushes, with simulated paint and virtual canvas.
- A physically-based interface that enhances the potential for transfer of real-world physical skills to the computer painting task.
- An interface that requires minimal training and is instantly accessible to users.
- An interface that allows for users to improve their skills over time, to become virtuosos.
- An interface with a novel use of haptic feedback, using two different models, to enhance the sense of immersion in the painting environment and increase the user's control of the brush.

Chapter 9

Conclusions

Cover the canvas at the first go, and then work on till you see nothing more to add ... Don't proceed according to rules and principles, but paint what you observe and feel. Paint generously and unhesitatingly, for it is best not to lose the first impression.
—Camille Pissaro

9.1 Summary of Results

In this dissertation I have presented a variety of physically-based techniques for modeling the tools and materials of traditional painting. The goal was to create a flexible, versatile, and easy-to-use digital painting system that works like painting in the real world, but with all the advantages of a digital system. In particular, I have presented a model for 3D flexible, dynamic paint brushes, three different models for the behavior and dynamics of paint, a realistic model for paint appearance, two models for haptic feedback, and an interface to tie all of these components together.

The brush model I presented, vBRUSH, uses quasi-static optimization of energy to stably simulate the stiff paintbrushes used in Western painting. The simulation is performed for a small number of brush spines that act like the skeletons used in skin-and-bones computer animation to deform the actual brush geometry. The hybrid geometric model combines smooth Catmull-Clark subdivision surfaces with hundreds of individually modeled bristles allowing for a wide variety of simulated brushes that can create a wide variety of marks. The modeling of brushes is further facilitated by the custom exporter I provide for creating brush models using the popular 3D modeling program, Blender (Blender, 2004). The many paintings shown in Chapters 3–6 clearly demonstrate the versatility of the vBRUSH brush model. I have also presented techniques for computing the contacting regions of the canvas and the brush textures, and for transferring paint between the two to effect a full bi-directional transfer of paint.

The several paint models I presented offer the user a trade-off between speed and physical fidelity. The simple two-layer dAb paint model offers the greatest interactivity on the widest range of hardware. It consists of one active, wet layer and one inactive, dry layer. The paint on each cell of each layer is stored as an RGB color and an 8-bit volume. Having all data in eight bit-per-component textures makes for efficient manipulation on a wide variety of graphics hardware. The Stokes Paint model is based on the 3D incompressible Stokes partial differential equations. The paint is stored on a 3D grid, and all layers can be active simultaneously. Finally the IMPaSTo model represents a balance between the speed and simplicity of dAb and the fidelity of partial differential equations in Stokes Paint. IMPaSTo represents paint using a stack of many dry layers, and a single wet, active layer. The dynamics model is based on five basic empirically derived physical principles of paint behavior, and these dynamics rules can be seen as simplifications of the Stokes equations used in Stokes Paint. In all of the paint models, for efficiency, interactions with the brush are carried out in a sliding window, only in the immediate vicinity of the brush.

The paint rendering model provides an unprecedented level of accuracy in the interactive simulation of paint mixtures and glazes. Using the Kubelka-Munk diffuse reflectance equations sampled at eight wavelengths, which are selected automatically via a Gaussian quadrature rule, and thanks to a fast GPU implementation, the rendering algorithm is able to transform full-spectrum measurements of real oil-paint samples into RGB colors for display in real time. The resulting colors calculated with only eight samples and Gaussian quadrature are almost indistinguishable from those calculated using a full 101 wavelengths and simple Riemann summation, at a vastly reduced computational cost. Furthermore the results are markedly improved compared with those obtained using just three samples of the spectrum with Kubelka-Munk, as previous work has done.

The haptic models give the painter the sensation of holding a real brush, as opposed to a pen or a mouse, and provide the tactile cues necessary to better control the brush. I presented a simple and fast haptic model based on one dimensional spring models, and also a more sophisticated model, which is the first method ever, to my knowledge, for generating haptic feedback directly from the velocity and pressure fields that are the output of a grid-based fluid simulation.

Although any one of the above components alone could probably have been pursued in greater depth as the subject of a dissertation, I chose to pursue all of them at once in the belief that the whole is greater than the sum of the parts. For example, a realistic, deformable brush model is not very compelling if the paint does not react realistically in response, and a sophisticated paint model without a realistic brush to manipulate it would make for an artistic tool with limited utility.

Though I have not conducted a formal user study, each of my models for brush and paint has been given extensive testing in the form of actual usage by painters with a wide variety of skill levels. Hundreds of people, both young and old, men and women, artists and non-artists, and from all over the globe, have tried my painting systems. Generally, I need only instruct them briefly to use the brush like they would a real brush, and to hit the space bar (or wave off the screen in that direction) when they want to use the palette. Sometimes when using the haptic input device I need to remind them that they can twist the brush, too, because people tend not to expect that to be possible based on their previous experience with computer input devices. They are generally then able to start making paintings right away. Many people complain, however, that they are not good artists, so they do not know how paint anything. Being able to teach these people to overcome their fear of painting is one of the great potential benefits of this type of system, because they can practice as much as they like without having to worry about wasting anything. And the uncluttered interface also allows them to focus completely on the task of painting.

As Pissaro exhorts us in the quote that begins this chapter, "Paint generously and unhesitatingly, for it is best not to lose the first impression". But typical computer painting interfaces often interfere with this directive, requiring the painter to switch tools or colors often, via invocations of a series of cumbersome menus. Part of the reason for needing frequent tool switching is the lack of versatility in the tool models themselves. A real painter does not need more than a handful of brushes because a single real brush can make a wide variety of different marks depending on pressure and loading, and on the 6DOF motion of the artist's hand. However, even the pressure-sensitive tools provided by the more sophisticated painting programs available today can usually at most vary a single parameter (e.g. size or opacity) in response to the user's input. The result is that painting programs end up requiring dozens of tools, each in dozens of different sizes. A real brush has greater flexibility and versatility, and the virtual 3D brush model I have presented brings these benefits to the world of digital painting, as well.

9.1.1 Mayer's Attributes

In the introduction I presented Mayer's list of the positive attributes of oil paints and brought attention to three in particular important for digital painting as well:

1. Its great flexibility and ease of manipulation, and the wide range of varied effects that can be produced.

2. The artist's freedom to combine transparent and opaque effects, glaze and body color, in full range in the same painting.

3. The dispatch with which a number of effects can be obtained by a direct, simple technique.

Each of these attributes is captured or reproduced in one way or another by the physically-based models for the traditional painting materials and tools that I have presented. In my systems, the last is perhaps due mostly to the versatile brush models; the second is most attributable to the paint models; and the first is achieved by equal parts of the two.

9.1.2 Process, Accident, and Complexity

In the introduction, I also discussed the three main qualities overlooked in most computer painting programs today: process, accident, and complexity. The overall design of the interface I have presented, which consists almost entirely of just brushes, palette, canvas and paint, ensures that the process of painting is the focus. Users cannot get their attention diverted searching for how to get some particular feature to work, because the features are all evident, and the features all lead the artist back to the central process of creating a work of art.

The physically-based models I have presented embody the rich, complex behavior of real natural paint media and tools. The 6DOF brush model with dynamic deformation enables the painter the create a wide variety of complex marks simply by varying how the brush is wielded. The paint can be spread, pushed, scraped and squished and it will react realistically, and not always predictably. Thanks to the organic complexity inherent in the tool models, the serendipitous accident, the “happy accident” as Bob Ross calls it, is not engineered out of existence in the pointless pursuit of precision.

9.2 Future Work

I have demonstrated several very capable physically-based models that are able to interactively deliver a physically realistic simulation of the behavior of painting materials and tools, and I have presented several painting systems based on these models to demonstrate their effectiveness. Nevertheless, there are still many exciting possibilities for improvements, and plenty of unexplored avenues for future research based on the successful results presented here.

First, I have already given a few specific suggestions for future improvements to specific components within the previous chapters covering each of models for brush, paint and haptics, so I will not repeat those here. But there are some future possibilities that have not been discussed relating to further applications of a 3D haptic painting system as a whole.

For instance, having a 6DOF input device and a haptic output device creates many new possibilities. It may be possible to use the haptic interface to record every motion made by a master painter and then draw on this motion library to help someone learning to paint by actually guiding their hands with haptic feedback. Or even aside from haptics it would be interesting to build up a model of a particular painter’s characteristic motions and technique. From this data one could possibly create new paintings in the same style automatically.

Another interesting possibility is stroking or fill by example. Currently with my systems, every single stroke must be painted by the user. But for covering a large area with small strokes, or with dots as in the *pointillism* style, this can be tedious.

Some painting programs feature modes or brushes that create particular instances of such effects, but it would be more useful if the user could demonstrate a short example of a stroking style, and then have the computer analyze and deduce the characteristic aspects of the motion. Then the user could fill large areas quickly with strokes characteristic of his or her own style. The computer's version of the motion should also, of course, display the same degree of variability that was in the human's motion. Then the human can simply guide the overall direction or perhaps the intensity of the motion via the input device in order to fill large areas with their own unique style, semi-automatically.

In terms of the materials and process supported by the painting simulation, there is much that can be done. There are many types of painting out there, and artists have a way of co-opting any material at hand to use in creating works of art. Bits of ribbon or hair, sand, paper scraps, or other random collage objects often find their way mixed in with the paint in modern works. Ideally such a wide variety of materials would be available to the digital artist as well. It would be interesting to create a whole separate program, still aimed at artists, but dedicated to designing materials for use in the digital artworks. These materials could behave in normal physical ways, like fluids or solids or anywhere in between, or they could behave differently from all known terrestrial materials, according to the artist's desire.

In terms of processes, Jackson Pollock created many of his paintings by dripping paint onto a large canvas on the floor as he walked around it. Another artist gained moderate fame in the 80's by creating paintings using buckets of paint blown onto a canvas by jet engines. These processes are outside the scope of possibility with my current models. What is required is some way to abstract the process at a higher level so as to encompass any process the artist can envision. Painting large scale works like murals requires a much different physical setup from that of smaller works. Scaling the painting system up to allow working on mural sized regions using projectors and tracking devices would be interesting.

An artist (a painter) on the radio recently said that he had realized that painting was about stillness, that that was its unique limitation that the painter must accept. But that is no longer necessarily true for paintings on a computer. There is a whole world of possibilities open when it comes to digital tools for creating moving, or interactive paintings. Paint can flow fluidly or form structures that morph into other structures. The great challenge is how to tame the complexity added by the extra dimension of time to create an interface that is not overwhelming.

I made extensive use of the computational power of GPUs in the IMPaSTo paint model. However, the speed and capabilities of GPUs continue to grow at an alarming rate. In the future it may be possible to perform all of the simulation on the GPU, brush, paint, and rendering, and with much higher resolution. For instance, in the near future the GPU may be able to simulate every one of the thousands of bristles on a real brush independently at interactive rates. Twenty years from now it will no doubt be possible to interactively perform a high fidelity simulation of a visco-elastic or visco-plastic paint-like material using a method like that in (Goktekin et al., 2004), which today requires a minute per time-step, and the GPU will likely be involved. In the future, even the haptic and friction models may be evaluated on the GPU, as the work of (Otaduy et al., 2004) indicates.

I have presented a natural, physical interface for one particular application, painting. However the use of physically-based user interface components could potentially lead to enhancements in a much wider range of applications. Certainly other content creation applications, such as 3D modeling and sculpting tools, are good candidates for a physically based interface, and some work has already been done in those areas (Foskey et al., 2002; FreeForm, 2004). In animation, interfaces for character posing currently use inverse kinematics extensively, but treating skeletons as elasto-plastic materials could make some tasks easier. In fact, any application in which a user works with virtual physical objects could benefit from incorporating the real-world physical behavior of those objects into the system via physically-based models.

This is of course not to say that physical models will *always* improve an interface. For instance, a word processor that models the motion of the arms of a mechanical typewriter smashing into the paper will not lead to a better word processor. But on the other hand, handwriting input systems can benefit from having a physical model of how pen and paper interact. Sousa's work on modeling pencils is another good example along the same lines (Sousa, 1999).

Even in non-critical capacities, physically-based modeling can add richness to a user interface. A good example is the digital audio mixing program, Reason, from Propellerhead Software (Reason, 2004). Reason presents the user with a virtual equipment rack. Turn the rack around and you can see all the cabling connecting various pieces of virtual audio equipment together. The cables are implemented using a dynamic, springy, physically-based model that allows the cables to bounce and drape naturally. The same function could have been accomplished by drawing line segments, but having physically simulated cables just adds a tangibility that encourages you to interact with the interface.

9.3 Conclusions

Painting is a form of art that has been practiced for centuries by people all over the world. Traditional painting will not ever disappear. But neither will digital painting. In this dissertation I have shown that it is possible to create realistic digital models of the primary tools in a traditional painter's toolbox, thereby creating a bridge between the traditional and the digital. A painting program enhanced with the physically-based brush, paint, and haptic models as I have presented here can provide the artist with a painting system that combines the best aspects of digital content creation with the many benefits of the traditional tools and processes. The result is a painting program that captures the "organic" feel of natural media desired by artists. It brings the three main advantages of oil painting cited by Mayer to bear, and it successfully injects the three important principles of process, accident, and complexity into the world of digital painting. Finally, the techniques I have presented in this dissertation enable painting systems that are both flexible and versatile while still being simple enough for a child to use; yet, like traditional painting, they offer a lifetime of potential to advance from novice to virtuoso.

Appendix A

Brush Optimization Derivations

Most optimization methods, including the SQP method I use for brush simulation, work best when at least first derivatives of the objective function and constraints can be computed. If they are very expensive to compute it is possible to estimate derivatives with finite differences, but this is not as reliable, and it requires multiple evaluations of the objective function to approximate one derivative. For brush energy optimization, the derivatives of Equation 3.1 with respect to all the joint parameters are needed, and these can all be expressed in closed-form. Computationally, the analytical derivative expressions are similar in cost to the objective function itself, and contain many common subexpressions, which can be reused for greater efficiency.

A.1 Spring Energy Derivative

The derivatives of the spring potential (Equation 3.2) for the Euler XYZ-angle representations are given by:

$$\frac{\partial}{\partial \theta_j} E_{s,i} = \frac{\partial}{\partial \theta_j} \frac{1}{2} K_i \beta_i^2 = 0, \quad \text{for } i \neq j \tag{A.1}$$

$$\frac{\partial}{\partial \phi_j} E_{s,i} = \frac{\partial}{\partial \phi_j} \frac{1}{2} K_i \beta_i^2 = 0, \quad \text{for } i \neq j \tag{A.2}$$

For $i = j$ one obtains:

$$\begin{aligned}
\frac{\partial}{\partial\theta_i}\frac{1}{2}K_i\beta_i^2 &= \frac{\partial}{\partial\theta_i}\frac{1}{2}K_i\left(\cos^{-1}(\cos\theta_i\cos\phi_i)\right)^2 \\
&= \frac{K_i\beta_i\sin\theta_i\cos\phi_i}{\sqrt{1-\cos^2\theta_i\cos^2\phi_i}}, \qquad \text{(A.3)} \\
\frac{\partial}{\partial\phi_i}\frac{1}{2}K_i\beta_i^2 &= \frac{K_i\beta_i\cos\theta_i\sin\phi_i}{\sqrt{1-\cos^2\theta_i\cos^2\phi_i}}. \qquad \text{(A.4)}
\end{aligned}$$

Equations A.3 and A.4 appear to be undefined at the brush rest pose, but they are actually both 0 in the limit.

A.2 Friction Derivative

For the derivatives of the isotropic friction potential (Equation 3.3), first I will introduce some notation for convenience. Let us denote the full vector of joint parameters simply as $\mathbf{q}$. Next, let $\mathbf{u} = \Delta\mathbf{x}_{c,i} = (\mathbf{p}_i - \mathbf{p}_i^0)$, where $\mathbf{p}_i = \mathbf{p}_i(\mathbf{q})$ is the current position of joint i, and $\mathbf{p}_i^0$ is its initial position. Differentiation with respect to the joint parameters gives us

$$\begin{aligned}
\frac{\partial E_{f,i}}{\partial\mathbf{q}} &= \frac{\partial}{\partial\mathbf{q}}\mu|F_{n,i}|\,\|\mathbf{u}\| \\
&= \frac{\partial}{\partial\mathbf{q}}\mu|F_{n,i}|\left((\mathbf{p}_i-\mathbf{p}_i^0)^T(\mathbf{p}_i-\mathbf{p}_i^0)\right)^{\frac{1}{2}} \\
&= \mu|F_{n,i}|\frac{(\mathbf{p}_i-\mathbf{p}_i^0)^T}{\|\mathbf{p}_i-\mathbf{p}_i^0\|}\frac{\partial\mathbf{p}_i}{\partial\mathbf{q}}, \qquad \text{(A.5)}
\end{aligned}$$

where $\partial\mathbf{p}_i/\partial\mathbf{q}$ is the Jacobian of the forward kinematics for joint i on the kinematic chain (Equation 3.11). The above is the friction derivative for a particular contact joint, i. The total friction derivative is the sum of all such terms, one for each joint of the brush skeleton that is in contact. The normal force, $|F_{n,i}|$, is assumed to be constant over the course of an optimization step.

A.3 Anisotropic Friction Derivative

Equation 3.14 can also be differentiated analytically by application of the product rule. To use the product rule the only additional information required is $\partial\eta/\partial\mathbf{q}$. Augmenting the notation above, let $\hat{\mathbf{u}} \equiv \mathbf{u}/\|\mathbf{u}\|$. If $\eta = 0$, I simply set $\partial\eta/\partial\mathbf{q} = 0$. For $\eta > 0$, one can show that

$$\frac{\partial\eta_i}{\partial\mathbf{q}} = C_\eta k(\mathbf{d}_p \cdot \hat{\mathbf{u}})^{k-1} \frac{\mathbf{d}_p^T}{\|\mathbf{u}\|}(\mathbf{I} - \hat{\mathbf{u}}\hat{\mathbf{u}}^T)\frac{\partial\mathbf{p}_i}{\partial\mathbf{q}} \tag{A.6}$$

then this can be combined with (A.5) using straighforward application of the product rule to get the full anisotropic $\partial E_f/\partial\mathbf{q}$. In the above, I have made use of the identity

$$\frac{\partial\hat{\mathbf{x}}}{\partial\mathbf{q}} = \frac{\partial\hat{\mathbf{x}}}{\partial\mathbf{x}}\frac{\partial\mathbf{x}}{\partial\mathbf{q}} = \left(\frac{\mathbf{I} - \hat{\mathbf{x}}\hat{\mathbf{x}}^T}{\|\mathbf{x}\|}\right)\frac{\partial\mathbf{x}}{\partial\mathbf{q}}. \tag{A.7}$$

For a concrete example, let us take $\mathbf{x} = \mathbf{x}(t)$, and calculate the rate of change of $\hat{\mathbf{x}}(t)$ with respect to time. Then one obtains

$$\dot{\hat{\mathbf{x}}} = \left(\frac{\mathbf{I} - \hat{\mathbf{x}}\hat{\mathbf{x}}^T}{\|\mathbf{x}\|}\right)\dot{\mathbf{x}}. \tag{A.8}$$

So, for example, if $\mathbf{x}$ is growing in the direction it already points, i.e. $\dot{\mathbf{x}} = c\mathbf{x}$, then one would expect from geometric principles that $\dot{\hat{\mathbf{x}}} = 0$. Substituting into Equation A.8,

$$\begin{aligned}
\dot{\hat{\mathbf{x}}} &= \left(\frac{\mathbf{I} - \hat{\mathbf{x}}\hat{\mathbf{x}}^T}{\|\mathbf{x}\|}\right)c\mathbf{x} \\
&= c\left(\mathbf{I} - \hat{\mathbf{x}}\hat{\mathbf{x}}^T\right)\hat{\mathbf{x}} \\
&= c\left(\hat{\mathbf{x}} - \hat{\mathbf{x}}(\hat{\mathbf{x}} \cdot \hat{\mathbf{x}})\right) \\
&= 0,
\end{aligned}$$

which is as expected.

A.4 Damping Derivative

The main complexities in differentiating the damping derivative are the inverse cosine hidden in the $\Delta\beta_i$ term, and the absolute value sign. Note, just as in Equations A.1–A.2, that for $i \neq j$, the derivative of the damping term is zero. For $i = j$ we have

$$\begin{aligned}
\frac{\partial \Delta\beta_i}{\partial \theta_i} &= \frac{\partial \beta_i}{\partial \theta_i} \\
&= \frac{\partial}{\partial \theta_i} \cos^{-1}(\cos\theta_i \cos\phi_i) \\
&= \frac{\sin\theta_i \cos\phi_i}{\sqrt{1-\cos^2\theta_i \cos^2\phi_i}} && \text{(A.9)} \\
\frac{\partial \Delta\beta_i}{\partial \phi_i} &= \frac{\cos\theta_i \sin\phi_i}{\sqrt{1-\cos^2\theta_i \cos^2\phi_i}}. && \text{(A.10)}
\end{aligned}$$

Given these, the full derivative can be computed as

$$\begin{aligned}
\frac{\partial E_{d,i}}{\partial \theta_i} &= \frac{\partial D_i |\Delta\beta_i|}{\partial \theta_i} \\
&= D_i \begin{cases} 0 & \text{if } \Delta\beta_i = 0 \\ -\partial\Delta\beta_i/\partial\theta_i & \text{if } \Delta\beta_i < 0 \\ \partial\Delta\beta_i/\partial\theta_i & \text{if } \Delta\beta_i > 0 \end{cases} && \text{(A.11)} \\
\frac{\partial E_{d,i}}{\partial \phi_i} &= D_i \begin{cases} 0 & \text{if } \Delta\beta_i = 0 \\ -\partial\Delta\beta_i/\partial\phi_i & \text{if } \Delta\beta_i < 0 \\ \partial\Delta\beta_i/\partial\phi_i & \text{if } \Delta\beta_i > 0 \end{cases} && \text{(A.12)}
\end{aligned}$$

Appendix B
An Artist's Statement

My View of IMPaSTo

John W. Holloway

6/5/04

I received a BFA in Painting from the University of Cincinnati in 1981. For the next 13 years I practiced the craft of painting on a full time basis in many of its different applications and permutations of technology, short that of digital. In 1995 I received an Associates of Science (Scientific Visulization/C++ programming) from Wake Tech Community college of Raliegh, NC. I then began work with what is now the "Technology Assisted Learning" (TAL) division of the "Research Triangle Institute International" (RTI) of Durham, NC.

My focus for the past 10 years has been the development of 3D environments for interactive real time training applications.

In the summer of 2001 I received a phone call from the floor of the Siggraph conference in Los Angeles from an associate. He described an experimental application being exhibited by Bill Baxter of UNC Chapel Hill, NC. My friend suggested that I look into Mr. Baxter's work due to my own interest in painting and digital graphics. I must admit I was skeptical at best.

I contacted Bill immediately and he was kind enough to permit me a couple of hours with his dAb, later to become IMPaSTo, application. In the course of those few hours I went from skeptic to true believer of Bills work and immediately recognized the possibilities and significance of this technology. For the past 3 years it has been my pleasure to experiment with IMPaSTo through its evolution.

There are many graphics applications available for image making and I have worked with a number of them. IMPaSTo offers a very direct bridge for a crossover in painting from the analog to the digital world. The haptic interface provides for the physical dimension and the use of volumetric modeling through voxels allows for the visual aspect, completing that circuit.

What is uniquely significant about IMPaSTo for me is the ability to approach digital image creation as a "painter" rather than a "graphic artist". This is a huge distinction. The IMPaSTo technology permits me to create a more pure "art" in the computer as opposed to creating pure "graphics" which is more the case with the dominant graphics development applications.

With this technology I am able to work with the freedom of the brush to canvass in a digital 3D environment. This freedom allows for the added pleasure of the unexpected, the "happy accident", that dynamic that makes for artful expression. The combination of interactions between the optical mixing of color, line, mark and texture that until now I could find only in the analog world I can now find in IMPaSTo. As this is a digital tool I am also provided the opportunity of versioning, not possible in analog painting. IMPaSTo also provides for the manipulation of some of the basic parameters to affect its behavior and manipulation of the digital medium.

Within the art world, traditional painters have rejected the computer primarily for its inability to produce images that are not purely graphic in nature. There is no "organicness", if you will, within digital images for obvious reasons. These images tend to be too clean, too sterile, too flat and too temporal. The need for an artist to evoke a feeling of soul is problematic and hardly fluid with the dominant digital technologies available. IMPaSTo represents a change in approach that begins to bridge this philosophical gap. IMPaSTo offers the introduction of, or rather, the simulation of that illusive "organic" factor.

I have found this technology to be extremely compelling. I am very grateful that Mr. Baxter has chosen to take this journey and has permitted me to tag along and gain a glimpse of what may prove to be a very significant development for the production of digital art.

It may be a reach to suggest that this technology could represent the beginning of a new genre in digital art creation and then, maybe not. This technology certainly contains the elements that such a movement could be built upon.

In this tool we may very well be witnessing the beginnings of a technology that will allow artists to create truly unique one of a kind digital artworks that make that final "fluid" connection between the analog and digital worlds.

Bibliography

Adams, B., Wicke, M., Dutré, P., Gross, M., Pauly, M., & Teschner, M. (2004). Interactive 3D painting on point-sampled objects. *Eurographics Symposium on Point-Based Graphics.*

Agrawal, S. K. & Fabien, B. C. (1999). *Optimization of Dynamic Systems.* Kluwer Press.

Agrawala, M., Beers, A. C., & Levoy, M. (1995). 3D painting on scanned surfaces. In Hanrahan, P. & Winget, J., editors, *1995 Symposium on Interactive 3D Graphics*, pages 145–150. ACM SIGGRAPH. ISBN 0-89791-736-7.

ArtRage (2004). Ambient Design. http://www.ambientdesign.com/artrage.html.

Ashton, D. (1972). *Picasso on Art: A Selection of Views.* Thames and Hudson.

Avila, R. S. & Sobierajski, L. M. (1996). A haptic interaction method for volume visualization. *Proceedings of Visualization'96*, pages 197–204.

Bandyopadhyay, D., Raskar, R., & Fuchs, H. (2001). Dynamic shader lamps : Painting on real objects. In *Proceedings of the International Symposium on Augmented Reality (ISAR 2001).*

Baxter, W. V. & Lin, M. C. (2004). A versatile interactive 3D brush model. *Proc. of Pacific Graphics 2004*, pages 319–328.

Baxter, W. V., Liu, Y., & Lin, M. C. (2004a). A viscous paint model for interactive applications. *Computer Animation and Virtual Worlds*, 15(3–4):433–442. http://gamma.cs.unc.edu/viscous.

Baxter, W. V., Scheib, V., & Lin, M. C. (2001). dAb: Interactive haptic painting with 3D virtual brushes. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 461–468. ACM Press / ACM SIGGRAPH.

Baxter, W. V., Wendt, J., & Lin, M. C. (2004b). IMPaSTo: A realistic model for paint. In *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering*, pages 45–56.

Bier, E. A., Stone, M. C., Fishkin, K., Buxton, W., & T.Baudel (1994). A taxonomy of see-through tools. *Proceedings of the ACM CHI'94 Conference on Human Factors in Computing Systems*, pages 358–364.

Blender (2004). Blender 3D. *http://www.blender3d.org.*

Bleser, T. W., Sibert, J. L., & McGee, J. P. (1988). Charcoal sketching: returning control to the artist. *ACM Trans. Graph.*, 7(1):76–81.

Bloomer, J. (2000). *Practical Fluid Mechanics for Engineering Applications.* Marcel Dekker.

Buxton, W. & Myers, B. A. (1986). A study in two-handed input. *Proceedings of the ACM CHI'86 Conference on Human Factors in Computing Systems*, pages 321–326.

Callet, P. (1996). Pertinent data for modelling pigmented materials in realistic rendering. *Computer Graphics Forum*, 15(2):119–127.

Carlson, M., Mucha, P. J., R. Brooks Van Horn, I., & Turk, G. (2002). Melting and flowing. In *Proceedings of the ACM SIGGRAPH symposium on Computer animation*, pages 167–174. ACM Press.

Catmull, E. (1974). *A subdivision algorithm for computer display of curved surfaces.* PhD thesis, University of Utah.

Chan, C. C. & Akleman, E. (2002). Two methods for creating chinese painting. *Proceedings of Pacific Graphics*, pages 403–412.

Chen, J. & Lobo, N. (1995). Toward interactive-rate simulation of fluids with moving obstacles using navier-stokes equations. *Graphical Models and Image Processing*, pages 107–116.

Chipp, H. B. (1968). *Theories of Modern Art.* University of California Press.

Chorin, A. (1968). Numerical solution of the navier-stokes equations. *Math. Comp.*, 22:745–762.

Chu, N. S. & Tai, C. L. (2002). An efficient brush model for physically-based 3D painting. *Proc. of Pacific Graphics*, pages 413–423.

Chu, N. S. & Tai, C. L. (2004). Real-time painting with an expressive virtual chinese brush. *IEEE Computer Graphics and Applications*, 24(5):76–85.

Cockshott, T. (1991). *Wet and Sticky: A novel model for computer based painting.* Ph.D. Thesis, University of Glasgow, Glasgow, Scotland.

Cockshott, T., Patterson, J., & England, D. (1992). Modelling the texture of paint. *Computer Graphics Forum (Eurographics'92 Proc.)*, 11(3):C217–C226.

Colgate, J. E. & Brown, J. M. (1994). Factors affecting the z-width of a haptic display. *IEEE Conference on Robotics and Automation*, pages 3205–3210.

Curtis, C., Anderson, S., Seims, J., Fleischer, K., & Salesin, D. (1997). Computer-generated watercolor. *Proc. of SIGGRAPH'97*, pages 421–430.

Deep Paint (2000). Right Hemisphere. *http://www.righthemisphere.com/products/dpaint/*.

Deep Paint 3D (2000). Right Hemisphere. *http://www.righthemisphere.com/products/dp3d/*.

Desbrun, M. & Cani, M. P. (1996). Smoothed particles: A new paradigm for animating highly deformable bodies. In *Computer Animation and Simulation '96 (Proceedings of EG Workshop on Animation and Simulation)*, pages 61–76. Springer-Verlag.

Dorsey, J. & Hanrahan, P. (1996). Modeling and rendering of metallic patinas. In Rushmeier, H., editor, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 387–396. ACM SIGGRAPH, Addison Wesley. held in New Orleans, Louisiana, 04-09 August 1996.

Duncan, D. R. (1940). The colour of pigment mixtures. *Proceedings of the Physical Society, London*, 52:380–390.

Duncan, D. R. (1949). The colour of pigment mixtures. *Journal of the Oil and Colour Chemists Association*, 32(7):269–321.

Duncan, D. R. (1962). The identification and estimation of pigments in pigmented compositions by reflectance spectrophotometry. *Journal of the Oil and Colour Chemists Association*, 45(5):300–324.

Duntley, S. Q. (1942). The optical properties of diffusing materials. *Journal of the Optical Society of America*, 32(2):61–70.

Durbeck, L., Macias, N., Weinstein, D., Johnson, C., & Hollerbach, J. (1998). Scirun haptic display for scientific visualization. *Phantom Users Group Meetings*.

Em, D. (1983). Butterfly nets revisited: The artist in the lab. In *ACM SIGGRAPH 1983*.

Enright, D., Marschner, S., & Fedkiw, R. (2002). Animation and rendering of complex water surfaces. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 736–744. ACM Press.

Evans, R. D. (1948). *An Introduction to Color*. John Wiley and Sons, Inc.

Expression 3 (2004). Creature House. http://www.microsoft.com/products/expression/.

Fedkiw, R., Stam, J., & Jensen, H. W. (2001). Visual simulation of smoke. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 15–22. ACM Press / ACM SIGGRAPH.

Fishkin, K. (1983). *Applying Color Science to Computer Graphics*. Master's thesis, University of California, Berkeley, CA.

Foskey, M., Otaduy, M., & Lin, M. (2002). Artnova: Touch-enabled 3d model design. *Proc. of IEEE Virtual Reality Conference.*

Foster, N. & Fedkiw, R. (2001). Practical animations of liquids. In Fiume, E., editor, *SIGGRAPH 2001, Computer Graphics Proceedings*, pages 23–30. ACM Press / ACM SIGGRAPH.

Foster, N. & Metaxas, D. (1996). Realistic animation of liquids. *Graphical models and image processing: GMIP*, 58(5):471–483.

FreeForm (2004). SensAble Technologies, Inc. http://www.SensAble.com.

Gair, A. (1997). *The Beginner's Guide, Oil Painting.* New Holland Publishers.

Gibson, S. (1995). Beyond volume rendering: Visualization, haptic exploration, and physical modeling of element-based objects. In *Proc. Eurographics workshop on Visualization in Scientific Computing*, pages 10–24.

Gleicher, M. (1998). Retargetting motion to new characters. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 33–42. ACM Press.

Goktekin, T. G., Bargteil, A. W., & O'Brien, J. F. (2004). A method for animating viscoelastic fluids. In *Proceedings of ACM SIGGRAPH 2004*. ACM Press.

Golub, G. H. & Van Loan, C. F. (1983). *Matrix Computations.* The Johns Hopkins University Press.

Gooch, B. & Gooch, A. A. (2001). *Non-Photorealistic Rendering.* AK Peters, Ltd.

Greene, R. (1985). The drawing prism: a versatile graphic input device. In *Proceedings of the 12th annual conference on Computer graphics and interactive techniques*, pages 103–110. ACM Press.

Gregory, A., Ehmann, S., & Lin, M. C. (2000a). *inTouch*: Interactive multiresolution modeling and 3d painting with a haptic interface. *Proc. of IEEE VR Conference.*

Gregory, A., Mascarenhas, A., Ehmann, S., Lin, M. C., & Manocha, D. (2000b). 6-dof haptic display of polygonal models. *Proc. of IEEE Visualization Conference.*

Griebel, M., Dornseifer, T., & Neunhoeffer, T. (1990). *Numerical Simulation in Fluid Dynamics: A Practical Introduction.* SIAM Monographcs on Mathematical Modeling and Computation. SIAM.

Guo, Q. & Kunii, T. L. (1991). Modeling the diffuse paintings of sumie. In *Modeling in Computer Graphics. Proc. of the IFIP WG 5.10 Working Conference*, pages 329–338.

Guo, Q. & Kunii, T. L. (2003). "nijimi" rendering algorithm for creating quality black ink paintings. *Proceedings of Computer Graphics International*, pages 152–159.

Haase, C. S. & Meyer, G. W. (1992). Modeling pigmented materials for realistic image synthesis. *ACM Trans. on Graphics*, 11(4):305.

Hall, R. (1989). *Illumination and Color in Computer Generated Imagery.* Springer Verlag.

Hanrahan, P. & Haeberli, P. E. (1990). Direct WYSIWYG painting and texturing on 3D shapes. In Baskett, F., editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 215–223.

Hapke, B. (1993). *Theory of Reflectance and Emittance Spectroscopy.* Cambridge University Press.

Harlow, F. H. & Welch, J. E. (1965). Numerical calculation of time-dependent viscous incompressible flow of fluid with free surface. *The Physics of Fluids*, 8(12):2182–2189.

Hays, J. & Essa, I. (2004). Image and video-based painterly animation. In *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering*, pages 45–56.

Hayward, V., Astley, O. R., Cruz-Hernandez, M., Grant, D., & Robles-De-La-Torre, G. (2004). Haptic interfaces and devices. *Sensor Review*, 24(1):16–29.

Helman, J. & Hesselink, L. (1990). Representation and display of vector field topology in fluid flow data sets. *Visualization in scientific computing*, pages 61–73.

Helman, J. L. & Hesslink, L. (1991). Surface representations of two- and three-dimensional F luid flow topology. In *Visualization '91*, pages 6–13.

Hertzmann, A. (1998). Painterly rendering with curved brush strokes of multiple sizes. *Proc. of ACM SIGGRAPH'98*, pages 453–460.

Hertzmann, A. (2001). Paint by relaxation. *Proc. Computer Graphics International*, pages 47–54.

Hertzmann, A. (2002). Fast paint texture. *NPAR 2002: ACM Symposium on Non-Photorealistic Animation and Rendering*, pages 91–96.

Hinsinger, D., Neyret, F., & Cani, M. (2002). Interactive animation of ocean waves. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation.*

Hirt, C. W. & Nichols, B. D. (1981). Volume of fluid (vof) method for the dynamics of free boundaries. *Journal of Computational Physics*, 39(1):201–225.

Hirt, C. W. & Shannon, J. P. (1968). Surface stress conditions for incompressible-flow calculations. *Journal of Computational Physics*, 2:403–411.

Hollerbach, J., Cohen, E., Thompson, W., Freier, R., Johnson, D., Nahvi, A., Nelson, D., Thompson, T., & Jacobsen, S. (1997). Haptic interfacing for virtual prototyping of mechanical cad designs. In *ASME Design for Manufacturing Symposium*.

Honour, H. & Fleming, J. (1995). *The Visual Arts: A History*. Harry N. Abrams, Inc.

House, D. & Breen, D., editors (2000). *Cloth Modeling and Animation*. AK Peters.

Hsu, S. C. & Lee, I. H. H. (1994). Drawing and animation using skeletal strokes. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 109–118. ACM Press.

Hsu, S. C., Lee, I. H. H., & Wiseman, N. E. (1993). Skeletal strokes. In *Proceedings of the 6th annual ACM symposium on User interface software and technology*, pages 197–206. ACM Press.

Huang, S.-W., Way, D.-L., & Shih, Z.-C. (2003). Physical-based model of ink diffusion in chinese ink paintings. *Journal of WSCG*.

Iwata, H. (1990). Artificial reality with force-feedback: Development of desktop virtual space with compact master manipulator. In Baskett, F., editor, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24, pages 165–170.

Iwata, H. & Noma, N. (1993). Volume haptization. *Proc. of IEEE VRAIS*, pages 16–23.

Jensen, H. W., Marschner, S. R., Levoy, M., & Hanrahan, P. (2001). A practical model for subsurface light transport. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 511–518. ACM Press.

Johnson, D., Thompson II, T. V., Kaplan, M., Nelson, D., & Cohen, E. (1999). Painting textures with a haptic interface. *Proceedings of IEEE Virtual Reality Conference*.

Johnson, G. M. & Fairchild, M. D. (1999). Full-spectral color calculations in realistic image synthesis. *IEEE Computer Graphics & Applications*, 19(4).

Johnson, P. (2003). *Art: A New History*. Harper Collins.

Johnston-Feller, R. (2001). *Color Science in the Examination of Museum Objects*. The Getty Conservation Institute.

Judd, D. B. & Wyszecki, G. (1963). *Color in Business, Science, and Industry*. John Wiley and Sons, Inc., 2nd edition.

Kabbash, P., Buxton, W., & Sellen, A. (1994). Two-handed input in a compound task. *Proceedings of the CHI'94 Conference on Human Factors in Computing Systems*, pages 417–423.

Kabbash, P., MacKenzie, I. S., & Buxton, W. (1993). Human performance using computer input devices in the preferred and non-preferred hands. *Proceedings of ACM InterCHI '93*, pages 474–481.

Kass, M. & Miller, G. (1990). Rapid, stable fluid dynamics for computer graphics. In *Proceedings of the ACM SIGGRAPH symposium on Computer animation*, pages 49–57. ACM Press.

Kershaw, D. S. (1978). The incomplete cholesky–conjugate gradient method for the iterative solution of systems of linear equations. *Journal of Computational Physics*, 26:43–65.

Kim, Y., Otaduy, M., Lin, M., & Manocha, D. (2002). 6-dof haptic display using localized contact computations. *Proc. of Haptics Symposium*, pages 209–216.

Körtum, G. (1969). *Reflectance Spectroscopy.* Springer-Verlag.

Kubelka, P. (1948). New contributions to the optics of intensely light-scattering material, part I. *J. Optical Society*, 38:448.

Kubelka, P. (1954). New contributions to the optics of intensely light-scattering material, part II: Non-homogenous layers. *J. Optical Society*, 44:p.330.

Kubelka, P. & Munk, F. (1931). Ein beitrag zur optik der farbanstriche. *Z. tech Physik*, 12:593.

Kunii, T., Nosovskij, G., & Hayashi, T. (1995). A diffusion model for computer animation of diffuse ink painting. In *Computer Animation '95*, pages 98–102.

Laerhoven, T. V., Liesenborgs, J., & Reeth, F. V. (2004a). A paper model for real-time watercolor simulation. Technical Report TR-LUC-EDM-0403, EDM/LUC. http://research.edm.luc.ac.be/ tvanlaerhoven/publications/vanlaerhoven_liesenborgs_tr03.pdf.

Laerhoven, T. V., Liesenborgs, J., & Reeth, F. V. (2004b). Real-time watercolor painting on a distributed paper model. In *Proceedings of Computer Graphics International.*

Landau, L. & Lifschitz, E. (1975). *Fluid Mechanics.* Pergamon Press.

Lawrence, D. A., Lee, C. D., Pao, L. Y., & Novoselov, R. Y. (2000). Shock and vortex visualization using a combined visual/haptic interface. *Proc. of IEEE Visualization*, pages 131–137.

Lee, J. (1999). Simulating oriental black-ink painting. *IEEE Computer Graphics & Applications*, 19(3):74–81.

Lee, J. (2001). Diffusion rendering of black ink paintings using new paper and ink models. *Computers & Graphics*, 25:295–308.

LeVeque, R. J. (1992). *Numerical Methods for Conservation Laws*. Birkhauser Verlag.

Lewis, J. P. (1984). Texture synthesis for digital painting. *Computer Graphics*, 18(3):245–252.

Lin, W.-J. & Shih, Z.-C. (2004). Computer-generated chinese painting with physically-based ink and color diffusion. *Proceedings of Computer Graphics Workshop*.

Litwinowicz, P. (1997). Processing images and video for an impressionist effect. *Proc. of SIGGRAPH'97*, pages 407–414.

Massie, T. M. & Salisbury, J. K. (1994). The phantom haptic interface: A device for probing virtual objects. *Proc. of ASME Haptic Interfaces for Virtual Environment and Teleoperator Systems*, 1:295–301.

Mayer, R. (1991). *The Artist's Handbook of Materials and Techniques*. Viking Books, 5 edition.

McNeely, W., Puterbaugh, K., & Troy, J. (1999). Six degree-of-freedom haptic rendering using voxel sampling. *Proc. of ACM SIGGRAPH*, pages 401–408.

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

Meriam, J. L. & Kraige, L. G. (1992). *Engineering Mechanics: Statics*. John Wiley & Sons,Inc., 3rd edition.

Meyer, G. W. (1988). Wavelength selection for synthetic image generation. *CVGIP*, 41:57–79.

Mi, X.-F., Tang, M., & Dong, J.-X. (2004). Droplet: A virtual brush model to simulate chinese calligraphy and painting. *Journal of Computer Science and Technology*, pages 393–404.

Muller, M., Charypar, D., & Gross, M. (2003). Particle-based fluid simulation for interactive applications. *Proc. of ACM SIGGRAPH/Eurographics Symposium on Computer Animation*.

Nahvi, A., Nelson, D., Hollerbach, J., & Johnson, D. (1998). Haptic manipulation of virtual mechanisms from mechanical CAD designs. In *Proc. of IEEE Conference on Robotics and Automation*, pages 375–380.

Nichols, B. D. & Hirt, C. W. (1971). Improved free surface boundary conditions for numerical incompressible-flow calculations. *Journal of Computational Physics*, 8:434–448.

O'Brien, J. F. & Hodgins, J. K. (1995). Dynamic simulation of splashing fluids. In *Computer Animation '95*, pages 198–205.

Oddy, R. J. & Willis, P. J. (1991). A physically based colour model. *Computer Graphics Forum (Eurographics)*, 10(2):121–127.

Osher, S. & Fedkiw, R. (2002). *Level Set Methods and Dynamic Implicit Surfaces.* Applied Mathematical Sciences. Springer-Verlag.

Otaduy, M. A., Jain, N., Sud, A., & Lin, M. C. (2004). Haptic display of interaction between textured models. *To appear in Proceedings of IEEE Visualization Conference. Austin, Tx.*

Otaduy, M. A. & Lin, M. C. (2003). Sensation preserving simplification for haptic rendering. *Proc. of ACM SIGGRAPH.*

Owen, R., Kurtenbach, G., Fitzmaurice, G., Baudel, T., & Buxton, W. (1998). Bimanual manipulation in a curve editing task. *Unpublished manuscript (http://www.billbuxton.com/CurveMatch.html).*

Painter 8 (2003). Corel. http://www.corel.com/painter/.

Pham, B. (1991). Expressive brush strokes. *CVGIP: Graph. Models Image Process.*, 53(1):1–6.

Photoshop (2004). Adobe. *http://www.adobe.com/photoshop/.*

Poirier, G. (2004). *Human Skin Modelling and Rendering.* Master's thesis, University of Waterloo. Technical Report Number CS-2004-05.

Posch, K. C. & Fellner, W. D. (1989). The circle-brush algorithm. *ACM Trans. Graph.*, 8(1):1–24.

Poynton, C. (2002). Color faq. http://www.poynton.com/ColorFAQ.html.

Reason (2004). Propellerhead Software. *http://www.propellerheads.se/products/reason/.*

Ristow, G. H. (1999). Particles moving in spatially bounded,viscous fluids. *Computer Physics Communications*, pages 43–52.

Ristow, G. H. (2000). Tumbling motion of elliptical particles in viscous two-dimensional fluids. *International Journal of Modern Physics C*, 11(0).

Rudolf, D., Mould, D., & Neufeld, E. (2003). Simulating wax crayons. In *Proc. of Pacifc Graphics*, pages 163–172.

Rudolf, D., Mould, D., & Neufeld, E. (2004). A bidirectional deposition model of wax crayons. *Conputer Graphics Forum.*

Ruspini, D., Kolarov, K., & Khatib, O. (1997). The haptic display of complex graphical environments. *Proc. of ACM SIGGRAPH*, pages 345–352.

Saito, S. & Nakajima, M. (1999). 3D physically based brush model for painting. *SIGGRAPH99 Conference Abstracts and Applications*, page 226.

Saito, S. & Nakajima, M. (2000). Physically based 3D brush model for interactive painting. *Jyouhou-Shori Gakkai Ronbunshi (A Japanese Journal)*, 41(3):608–615.

Saito, T. & Takahashi, T. (1990). Comprehensible rendering of 3D shapes. *Computer Graphics (SIGGRAPH'91 Proc.)*, 24(4):197–206.

Salisbury, K., Brock, D., Massie, T., Swarup, N., & Zilles, C. (1995a). Haptic rendering: Programming touch interaction with virtual objects. In Hanrahan, P. & Winget, J., editors, *1995 Symposium on Interactive 3D Graphics*, pages 123–130. ACM SIGGRAPH. ISBN 0-89791-736-7.

Salisbury, K., Brock, D., Massie, T., Swarup, N., & Zilles, C. (1995b). Haptic rendering: Programming touch interaction with virtual objects. *Proc. of 1995 ACM Symposium on Interactive 3D Graphics*, pages 123–130.

Schofield, S. (1994). *Non-photorealistic Rendering: A critical examination and proposed system.* PhD thesis, School of Art and Design, Middlesex University.

Schwartz, L. (1992). *The Computer Artist's Handbook: concepts, techniques, applications.* Computer Creations Corporation, 1st edition.

Searby, A. D. & Walker, I. C. (1985). Computerized graphics system and method using an electronically synthesized palette. United States Patent Number 4,524,421.

Shewchuk, J. (1994). An introduction to the conjugate gradient method without the agonizing pain. Technical Report CMUCS-TR-94-125, Carnegie Mellon University. (See also http://www.cs.cmu.edu/ quake-papers/painless-conjugate-gradient.ps.).

Shoup, R. (2001). Superpaint: An early frame buffer graphics system. *IEEE Annals of the History of Computing*, 23(2):32–37.

Small, D. (1991). Simulating watercolor by modeling diffusion, pigment, and paper fibers. In *SPIE Conference Proceedings*, volume 1460, pages 140–146.

Smith, A. R. (1978). Paint. TM 7, NYIT Computer Graphics Lab.

Smith, A. R. (1997). Digital paint systems: A historical overview. TM 14, Microsoft.

Smith, A. R. (2001). Digital paint systems: An anecdotal and historical overview. *IEEE Annals of the History of Computing*, 23(2).

Sourin, A. (2001). Functionally based virtual computer art. In *Proceedings of the 2001 symposium on Interactive 3D graphics*, pages 77–84. ACM Press.

Sousa, M. C. (1999). *Computer-Generated Graphite Pencil Materials and Rendering.* Ph.D. Thesis, University of Alberta, Alberta, Canada.

Stam, J. (1999). Stable fluids. In Rockwood, A., editor, *Siggraph 1999, Computer Graphics Proceedings*, pages 121–128, Los Angeles. Addison Wesley Longman.

Strassmann, S. (1986). Hairy brushes. *Computer Graphics (SIGGRAPH'86 Proc.)*, 20:225–232.

Strothotte, T. & Schlechtweg, S. (2002). *Non-Photorealistic Computer Graphics: Modeling, Rendering and Animation.* Morgan Kaufmann.

Subramanian, R. S. (2003). Non-newtonian fluids. From lecture notes for Fluid Mechanics, http://www.clarkson.edu/subramanian/ch301/notes/nonnewtonian.pdf.

Sutherland, I. (1963). *Sketchpad, A Man-Machine Graphical Communication.* Ph.D. Thesis, Massachusetts Institute of Technology, Cambridge, MA.

Thompson, T., Johnson, D., & Cohen, E. (1997). Direct haptic rendering of sculptured models. *Proc. of ACM Interactive 3D Graphics*, pages 167–176.

Tia (2001). Interview with Tia. *http://www.pixar.com/artistscorner/tia/interview.html.* Interview of a Pixar artist found on Pixar's web site.

Tome, M. F. & McKee, S. (1994). Gensmac: A computational marker and cell method for free surface flows in general domains. *J. Comp. Phys.*, 110:171–186.

Tu, X. (1996). *Artificial Animals for Computer Animation: Biomechanics, Locomotion, Perception, and Behavior.* PhD thesis, University of Toronto.

Ware, C. & Baxter, C. (1989). Bat brushes: on the uses of six position and orientation parameters in a paint program. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 155–160. ACM Press.

Warnick, K. F. (2001). Gaussian quadrature and iterative linear system solution methods. *http://www.ee.byu.edu/ee/class/ee563/notes/gq_tutorial.pdf".*

Whitted, T. (1983). Anti-aliased line drawing using brush extrusion. In *Proceedings of the 10th annual conference on Computer graphics and interactive techniques*, pages 151–156. ACM Press.

Williams, L. (1990). 3D paint. In *Proceedings of the 1990 symposium on Interactive 3D graphics*, pages 225–233. ACM Press.

Winkenbach, G. & Salesin, D. H. (1994). Computer-generated pen-and-ink illustration. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 91–100. ACM Press.

Winkenbach, G. & Salesin, D. H. (1996). Rendering parametric surfaces in pen and ink. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 469–476. ACM Press.

Witkin, A. & Baraff, D. (1997). *Physically Based Modeling: Principles and Practice.* ACM Press. Course Notes of ACM SIGGRAPH.

Witkin, A. & Kass, M. (1988). Spacetime constraints. In Dill, J., editor, *Computer Graphics (SIGGRAPH '88 Proceedings)*, volume 22, pages 159–168.

Wong, H. & Ip, H. (2000). Virtual brush: A model-based synthesis of chinese calligraphy. *Computers & Graphics*, 24.

Wright, W. D. (1958). *The Measurement of Colour.* The MacMillan Company.

Wyszecki, G. & Stile, M. (1982). *Color Science.* Wiley.

Wyvill, B., van Overveld, K., & Carpendale, S. (2004). Rendering cracks in batik. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering*, pages 61–69. ACM Press.

Xu, S., Lau, F., Tang, F., & Pan, Y. (2003). Advanced design for a realistic virtual brush. *Computer Graphics Forum (Eurographics '03)*, 22(3):533–542.

Xu, S., Tang, F., Lau, F., & Pan, Y. (2002). A solid model based virtual hairy brush. *Computer Graphics Forum (Eurographics '02)*, 21(3):299–308.

Yeh, J.-S., Lien, T.-Y., & Ouhyoung, M. (2002). On the effects of haptic display in brush and ink simulation for chinese painting and calligraphy. *Proc. of Pacific Graphics 2002 (PG2002)*, pages 439–441.

Yu, Y. J., Lee, D. H., Lee, Y. B., & Cho, H. G. (2003). Interactive rendering technique for realistic oriental painting. *Journal of WSCG*, 11(1):538–545.

Yu, Y. J., Lee, Y. B., Cho, H. G., & Lee, D. H. (2002). A model based technique for realistic oriental painting. In *10th Pacific Conference on Computer Graphics and Applications (PG'02)*, pages 452–453.

Z-Brush (2000). Pixologic. *http://pixologic.com.*

Zhang, Q., Sato, Y., Takahashi, J., Muraoka, K., & Chiba, N. (1999). Simple cellular automaton-based simulation of ink behaviour and its application to suibokuga-like 3D rendering of trees. *Journal of Visualization and Computer Animation*, 10:15–26.

Zimmer, M. A. (1994). System and method for digital rendering of images and printed articulation. United States Patent Number 5,347,620.

Zimmer, M. A. (1998). Digital mark-making method. United States Patent Number 5,767,860.