Making Digital Painting Organic

by

CHU SIU HANG

A Thesis Submitted to

The Hong Kong University of Science and Technology

in Partial Fulfillment of the Requirements for

the Degree of Doctor of Philosophy

in the Department of Computer Science and Engineering

August 2007, Hong Kong

Authorization

I hereby declare that I am the sole author of the thesis.

I authorize the Hong Kong University of Science and Technology to lend this thesis to other institutions or individuals for the purpose of scholarly research.

I further authorize the Hong Kong University of Science and Technology to reproduce the thesis by photocopying or by other means, in total or in part, at the request of other institutions or individuals for the purpose of scholarly research.

_____

CHU SIU HANG

Making Digital Painting Organic

By

CHU SIU HANG

This is to certify that I have examined the above PhD thesis
and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by
the thesis examination committee have been made.

_____

Dr. CHIEW-LAN TAI, Thesis Supervisor

_____

Prof. LIONEL NI, Head of Department

Department of Computer Science and Engineering

30 August 2007

Making Digital Painting Organic

By CHU SIU HANG

Department of Computer Science and Engineering

The Hong Kong University of Science and Technology

Abstract

In this thesis I present physically-based techniques for building real-time digital painting tools. In particular, I focus on two components that are most needed in existing paint systems: 3D brush modeling and water-based paint simulation. A 3D deformable brush model is crucial in generating organic brush strokes. When paired with a proper input device, such a brush model also allows true-to-life control of stroke creation because users can see how the brush deforms in response to their manipulation while painting.

Existing paint programs do simulate various paint media including charcoal, oil, and watercolor. However, water-based media like Eastern ink and Western watercolor are the most unsatisfactory due to their inability to produce natural-looking marks, especially those with flow effects. In the second part of my thesis, I present a technique based on a relatively new computational fluid dynamics method called Lattice Boltzmann to simulate water-based paint media. Many of the existing real-world artistic effects as well as novel ones are achievable with this approach.

Acknowledgements

I have been very fortunate. I live in a place without war, am well fed and have the opportunity to choose what I wanted to do. I was also lucky enough to be born in an Eastern culture, yet grew up in a place where East meets West. This allows me to see things from multiple perspectives.

Eight years ago I embarked on the journey of developing a 3D virtual brush model for digital paint applications. At first I just wanted to do brushes, but later I had the opportunity to work on ink flow simulation. This PhD thesis is a conclusion of what I have been working on for the past years.

Many people have helped along the way. I would like to first thank my supervisor Chiew-Lan Tai for her support and guidance all those years. It is a privilege to be given the opportunity to work on things that I like. I also thank my thesis examination committee members, Prof. Tien-Tsin Wong, Prof. Long Quan, Dr. Kun Xu, and Dr. Philip Fu, for their careful reading and comments. I'm grateful to my labmates at VISGRAPH Lab for a friendly environment. Over the years, I witness the transition from local students to mainland Chinese being the majority in the lab. It was nice to learn from all of them and I did not miss the chance to practice my Mandarin with the mainlanders.

We are lucky enough to gain recognition from the industries and I would like to thank Tony Eastham and his colleagues for taking the administrative burden in our technology transfer. I need to thank people from the software and the hardware industries: Jerry Harris (Adobe Systems Inc.), the man behind the Photoshop paint engine, for bringing me aboard Adobe and his enthusiastic support; Julie Kmoch, John Worthington and John Nack (Adobe Systems Inc.) who facilitate the incorporation of technologies described in this thesis into Adobe's best-selling products; Ichinaka Yasuhiro (Sony Electronics Inc.) for bringing our technologies to their company and his keen support; Yoshihiro Yoshioka (Sony Electronics Inc.) for his help in porting code to Sony's upcoming Cell Computing Board and in finding good food in Japan.

Apart from real business, I also got involved in academic business plan competitions because of our MoXi paint system. I thank Tony again for introducing MoXi to some MBA students of our university who needed some technology to sell in their business plan. I also thank the MBA students and their Professors for their business advice and encouragements.

I‘m grateful to my fellow computer graphics researchers: Sylvain Paris for his advices and sharing; Xavier Grainer for reviewing our technical papers and encouragement; Alex Hsu, the original developer of Microsoft Expression, for giving advice at times I needed the most; Researchers at State University of New York at Stony Brook for introducing the Lattice Boltzmann Method to the computer graphics community. I thank artists in traditional and digital media: Ink painters and designers Wucius Wong and Tai-Keung Kan for their advice and encouragement; digital painters Paulo Purim, Jeff Wong and many others I met over the Net for their encouragement and sharing; digital painters Yoko Orge for writing a review of MoXi in an online magazine, which keep sending Japanese readers to our MoXi website; I thank people from the CG motion pictures industry: Mark Lam and Pak-Ho Wong (cgvisual.com) for their interview article which made us known to many of the Chinese computer graphics practitioners; Eddy Wong (Menfond Electronic Arts) for getting me involved in movie special effect production; Special effect artists Todd Akita (Psyop), Matt Estela (House of Curves), Miguel Campos (spiderland.tv) and Jono Cardno (Flux Animation) and Bun Kwai (ManyMany Creations) for taking the challenge of experimenting the MoXi prototype to produce special effects for real-world projects.

I thank people from the mass media, including Lai-Fan Ho (RTHK) and Natsuko Sotomura (mycom.co.jp) for reporting our story. Angelica Leung introduced me to local artists. Kwan-Wah Ng helped us with the initial brush tracking hardware. My research was not possible without the funding support from the HKSAR government and the SINO Group. With technology transfer going on since the past two years, I hope they agree that their money wasn’t wasted. Finally, I want to thank my family for their understanding and sacrifices.


香港
科技大學

TABLE OF CONTENTS

List of Figures

List of Tables

CHAPTER 1

INTRODUCTION

The computer is an extremely versatile tool for many tasks including artistic creation. Nowadays, many artists have adopted digital tools due to its possibilities not matched by conventional media. As an art tool, the computer also bears great potential on the development of new art forms and techniques simply not possible in the real world. Painting is one of the most appreciated and practiced art forms throughout human history. Today, this art has also been digitalized through the use of digital paint systems. Although digital painting has been around for over thirty-five years [Smith 2001], there are still several issues yet to be addressed. In this thesis, I focus on bringing the 'organicity' (and fun!) offered by real paint brushes and water-based painting media into the digital world.

1.1 Digital Painting

Today, digital painting is popular in applications ranging from conceptual design for film or computer games to illustration for books or magazines. Digital painter usually paints with a stylus on a graphics tablet (Figure 1.1), which acts like a pen on a drawing board[1]. Digital painting differs from other forms of computer-generated art in that the artwork is painted stroke by stroke manually instead of being rendered by some computer algorithms. Instead of pulling curves using a mouse, digital painting also keeps the 'manual' feel of the traditional craft.

[1] It is possible to paint with a mouse, but this would be as cumbersome as painting with a piece of brick.

Figure 1.1: A graphics tablet with a stylus.

One may ask why some artists prefer digital over traditional painting. The topmost reasons for going digital given by existing digital artists are:

- Able to undo and redo strokes;
- Able to save on materials expenses;
- Able to save artwork at different stages;
- Able to playback previously recorded strokes;
- Able to work with much more control or flexibility;
- No tedious setup and cleanup.

Perhaps the number one reason for artists to paint digitally is the ability to undo. The ability to undo strokes makes the artist feel free to make mistakes, thus encouraging them to experiment. When asked about the importance of undo, Paulo Purim, a digital painter from Brazil, said *'Undo changes everything about how things are done!'*

High-quality paints and papers are expensive. With digital paint, there is virtually no cost for the consumables, and artists no longer have to worry that certain colors being used up in the middle of a painting session. This is just like how the digital camera has allowed people to take casual snapshots without worrying about wasting films.

Figure 1.2: Left: Jeff Wong in front of his digital painting. Right: His painting used in the cover of a magazine.

Digital illustrator Jeff Wong did a magnificent digital painting based on the famous ceiling of the Sistine Chapel (Figure 1.2). When asked why he chose to do it digitally, he said it would be too cumbersome to work on a real canvas big enough for all those details. Working digitally just made this project more feasible.

1.2 Motivations

In this thesis, I present new technologies for brush and water-based paint simulation for digital artwork creation purposes. I have basically two motivations for my work. My primary motive is that existing paint systems just do not give what I, as an artist, wanted. As a computer user, I got my first graphics tablet in 1993. Soon afterwards, I realized that using my tablet with existing paint programs could never produce the kind of organic strokes that I could make with a real calligraphy brush. I felt the need for better digital brushes. I will elaborate on this later.

The other motivation was to embrace the power of computer to further develop art traditions. Progressive painters are constantly looking for new tools or styles. For specialists in traditional media, new digital effects are actually a much better reason to use a computer – the convenience listed in Section 1.1 might just be offset by the fact that it is hard for senior painters to operate a computer. Besides, I also consider further development of art forms using computer as an art in itself. This is challenging and I would like to participate in the campaign.

1.3 What Artists Want

As mentioned in the previous sub-section, it is precisely because existing systems do not give what I wanted that I embarked on the journey of digital art tool making. During the course of development, I received emails from all over the world concurring my feeling. For instance, Daniel Morkowski from France said,

> *"I find myself confronted, as you no doubt are aware, with the limited performances of the virtual tools currently available. Your research paper shows clearly that you understand only too well these limitations."*

There are certainly different types of artists wanting different things. Summarizing from communication with fellow artists and my own research, I now discuss in the following sub-sections exactly what artists want in their paint systems. Generally speaking, the first three sub-sections below represent different levels of requirements – the casual artists may just want organicity to get a good-looking outcome, while the professional artists may prefer to have more spontaneous control for better expression, and the most aggressive artists may want radically different tools for breakthroughs. Finally in the last sub-section, I also discuss the physicality of digital artwork, which is still quite difficult to get even with the latest hardware technology.

1.3.1 Organicity

Painter John W. Holloway, when commenting on existing paint systems, said [Baxter 2004],

> *"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."*

In deed, early paint systems were all designed to give marks that look very regular. This is partly due to the fact that computer power at that time simple doesn't allow complicated real-time simulations of various media. Another reason for not having

organic paint system might be that most software engineers still have the mindset that painting is simply all about dabbing colors onto a canvas without showing traces of the strokes as in the realistic painting style developed in the Renaissance (Figure 1.3 left). If we can have variable dab size, we can always paint every pixel the way we want, thus satisfying the carefully dabbing need for a realistic painting. However, in the recent history of Western painting, painters tend to show traces of brush strokes (Figure 1.3 right), instead of hiding them.

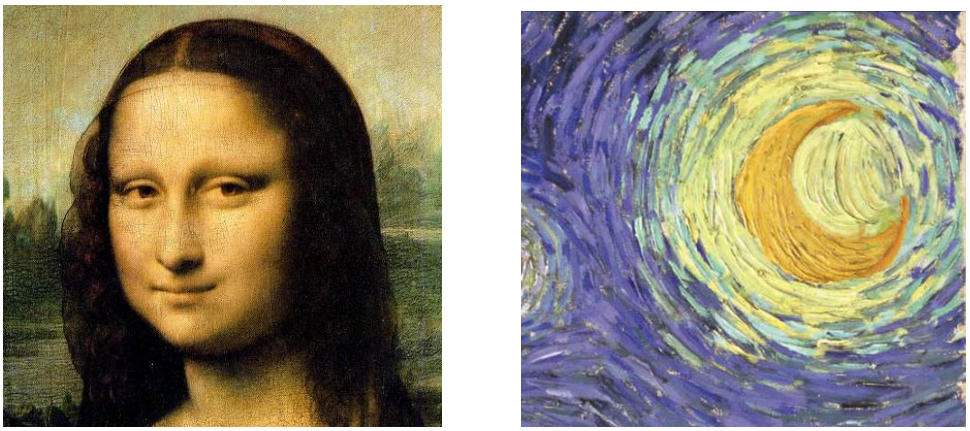


Figure 1.3: Sample oil paintings different painting styles. Left: Mona Lisa (part of) by Leonardo Da Vinci 1506. Right: *Starry Night* (part of) by Vincent Van Gogh 1889 (images from www.wikipedia.org).

It is worth noting that we use the term 'organic' rather than 'realistic' when we talk about a digital stroke. I take 'organic' to mean 'natural-looking'. Being able to simulate paint as realistic as possible is certainly a great engineering or scientific achievement, but organic strokes need not be the most realistic.

1.3.2 Spontaneity

Another aspect that is very much related to organicity is *spontaneity*. While organicity is more about the final appearance, spontaneity is about the process. It is possible to get organic strokes by modeling only the appearance [Hsu & Lee 1994], but do digital painters want to reach organic outcome via spontaneous painting? Hsu and Lee [1994] argue that if we are more interested in the final appearance, whether the strokes are painted with a simulated brush may not be important after all. In my opinion, it is

absolutely legitimate for people to care about the outcome more than the process. However, I believe we, as software tool makers, should also cater for people who actually care about the latter. I myself have experience in Chinese calligraphy and I can say at least such an art actually *depends* on brush wielding. I defer my explanation on why the process is important in Section 2.1. Apart from Chinese art, I can also quote some artist from the West. In 2002, when commenting on a future version of *Corel Painter* [Painter 2007], veteran digital painter Paulo Purim said,

> *"I want more bristles and better ones! I want split brushes and real-looking dry brushes. Do you know what happens when you press hard a dry brush on the canvas so that the bristles open in a flower-like thing? I want to do that. More than anything, I want bristles that get thicker when you resize the brush, so that they won't always look pixel-sized strings."*

This statement clearly shows that Paulo wanted the brush to deform more realistically as his paints so that he could control his brush just like in the real world – a request for a well simulated brush for the creative process.

Even for real painting media, the interest of using more spontaneity has been developed in the West in recently years. For example, William Alexander developed in the 60's a quick and direct wet-on-wet oil painting technique [Wikipedia Alexander 2007], in which paint is mixed right on the canvas. Referred as 'Happy Painter', William had many followers due to his television art shows. Bob Ross continued William's spontaneous tradition attracting even more appreciators [Wikipedia Ross 2007]. One central idea in Bob's painting is *joy*, which can be felt when he used 'happy accident' to describe unintended strokes painted spontaneously. The process of painting itself often brings joy. Another example is Donna Dewberry's *One Stroke* technique [OneStroke 2007], in which one paints with multiple colors loaded on a paintbrush and using very few strokes. In my opinion, Donna's technique resembles the spontaneous style of Chinese ink painting a lot, except her medium is acrylic instead of watercolor or ink.

Spontaneous painting often requires better brush manipulation skills. On the other hand, *graphics design* often involves more about manipulation of the final appearance

than how each element in the design is produced. In conclusion, whether an artist needs spontaneity really depends on what kind of art his or she is into and spontaneity should be provided for those who want it.

1.3.3 Development

Progressive painters are constantly looking for new artistic expression. In the West, artists like Henri Matisse (1869-1954) or Pablo Picasso (1881-1973) contributed a lot in modernizing Western painting by breaking well-set rules that dictated what is considered 'beautiful'. In the East, renowned painter Shi-Tao (石濤) from the 18th century, whose painting is considered exceeding his own generation, commented that *'Expression should follow with time'* (筆墨當隨時代). Modern painter Wu Guan-Zhong (吳冠中) also said, *'To keep the tradition is to develop the tradition*' (保留傳統，只有發展才能保留，不發展就不可能保留). We can see that these individuals have strong liberality to further develop the art. Today, such ideas are widely accepted and many artists would say they take an innovative route in their artists endeavor. But exactly how do they innovate?

Traditional-media painters have been trying hard finding new ways to use physical media for new effects. For example, Liu Guo-Song (劉國松) formulated his own painting paper, which allows him to tear away paper fibers to create white streaks in his ink painting. Traditional-media artists have asked questions like: how to give a sense of fluidity to oil painting? How to get strong colors in ink painting? Not limited by physical constraints, I believe that the computer opens up a whole new avenue for developing new tools that answer the many questions traditional artists have. Digital revolutions have already happened, for example, in the area of computer-generated animation. New artistic developments should just be continued.

1.3.4 Physicality

One of the artists I talked to in person is innovative ink painter Wucius Wong (王無邪). He has worked with traditional media for decades and started to work with computer since early 1990's. He commented that there are two things he does not like in digital art. One is that the artwork is *virtual* and there is nothing he could get to hold of in his

hands. The other is that graphics tablet stylus just does not give him the same level of touch feedback possible with real brushes. In fact, one of the main concerns traditional painters have about digital painting is the lack of correct force feedback. In Chinese painting and calligraphy, a piece of felt cloth is often placed underneath the painting paper so as to avoid excessive moisture from staining the paper. However, some calligraphers actually prefer to write on a hard table without the felt so that they could feel the force feedback better. We see how important the feel is.

1.4 Challenges

I would like to remind the reader that digital paint systems have been around for over 35 years but many artists are still not satisfied because the above-mentioned needs are still not fully met. To improve this situation, there are several challenges in bringing the good of traditional painting into the digital realm and further developing the art.

1.4.1 Physically-based Simulation

By 'physically-based simulation', I mean the simulation of virtual paint tools and media based on physics principle or laws. Although we should not limit ourselves to certain ways of making digital art tools, physical simulation is one of the main challenges digital art tool makers face today. One may ask why we should do complex physical simulation instead of simple imitation of brush and paint, or even just create completely new media specific to computer [Schofield 1994]. One reason is that, with tools that behave like objects in the real world, it is much more intuitive for artist to learn and expect the outcome. Many artists do not use the computer just because of steep learning curves. This is concurred by Bill Baxter in his PhD dissertation [Baxter 2004]. Another reason is that in Chinese painting and calligraphy, the brush and ink dynamics are actually important in determining the quality of the outcome.

Physical simulation is computationally intensive and it is not easy to produce a physically-based system with best user experience possible. Our challenge is to devise good simulation techniques that run fast enough for interactive painting at reasonable resolution.

1.4.2 Resolution

In current paint systems, it is possible to create a painting in a resolution as high as the system memory allows. Since most current digital painters work on matte painting for films or illustrations for books or magazines which require limited resolution, this is reasonable good enough. In fact, if even high resolution is needed, we can always divide the paintings into tiles, just like what photo-realist painter Bert Monroy (www.bertmonroy.com) does. This is why most existing digital artists do not complain about resolution.

Nevertheless, it is still desirable if we can zoom in the artwork indefinitely just as possible with real painting. This is no problem if the digital graphics is vector-based but if we are to do physical simulation we are mostly bitmap-bound [Wikipedia raster 2007] because most simulation algorithms can only be implemented on discrete image space. This presents researchers the problem of how to allow high resolution output comparable to real paintings of the size several square meters which we often find hanging in some big halls or living rooms.

1.4.3 Physical Artwork

The computer was not designed for painting originally and it is intrinsically difficult to get physicality for virtual art. For getting something he could get hold of, Wucius Wong solves, within the scope of ink painting, by printing artwork on real Chinese *Xuan* painting paper with an inkjet printer. Printer ink penetrates Eastern art paper much like real Chinese ink, giving the printout the look of conventional artwork. To make it more indistinguishable from real artwork, he would often add some real brush strokes to hide the edge of the printed image. As suggested by him, I also printed some artwork made with my own paint system and a few pieces are shown in Figure 1.4. I myself find the printing process as not very easy as the thin paper is easily jammed in the printer. I currently solve this problem by first mounting the paper on a piece of plastic sheet before feeding it to printer.

Figure 1.4: Artwork printed using an ink jet printer on thin Eastern painting paper.

Western people have developed *giclée*, which refers to the use of the ink-jet printing process for making fine art prints from digital sources [Wikipedia giclée 2007]. In fact, giclée printing is already quite popular in the West (Figure 1.5). Prints can also be made on watercolor paper for aesthetic look. To get a thick surface texture, it is also possible to print on canvas (Figure 1.6 left). However, getting a thick paint feel of real impasto paintings is still difficult. Rapid prototyping machines or 3D printers [Wikipedia 3DPrint 2007] are available, but they are geared towards making 3D models rather than painting. Figure 1.6 (right) shows a sample artwork made using Computer Numerically Controlled (CNC) prototyping processes for relief effect.

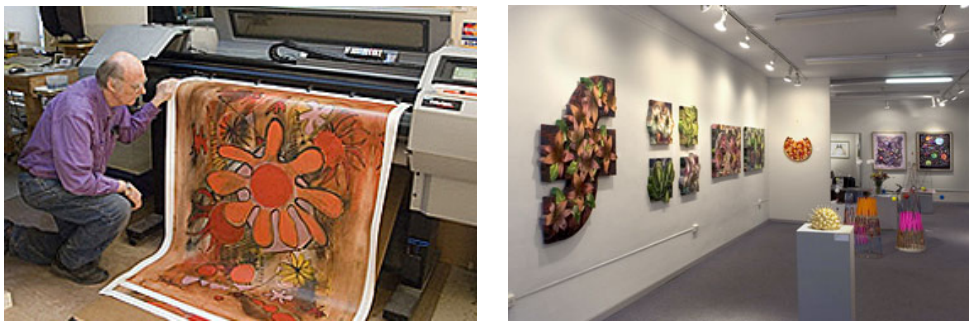


Figure 1.5: Left: Stan Bowman making giclée print. Right: His works shown at a gallery (images from www.stanbowman.com).

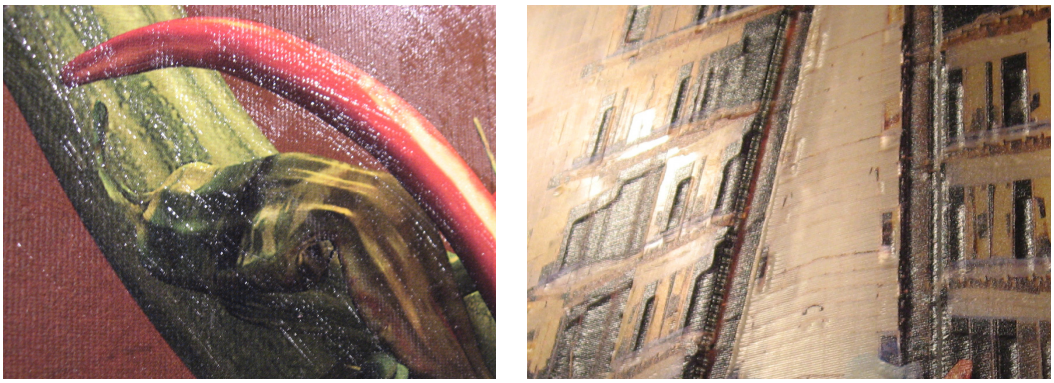


Figure 1.6: Left: Close-up of a giclée print by Till Nowak, 2006. Right: Close-up of a photograph etched on lexiglass by Murat Germen 2007.

1.4.4 Touch

The last challenge I would like to mention is how to give proper force feedback. There are six-degree-of-freedom haptic device and 3D spatial and orientation sensors available, but in my opinion, there is still no perfect solution if we are to give both the free movement and the force feedback a real brush offers to the digital brush user. A discussion of existing input devices will be presented in Section 5.2.

1.5 Thesis

My thesis statement can be formulated as follows:

> *By devising efficient simulation methods that capture the complex behaviors of real brushes and water-based paint, one can create an intuitive paint system that allows the creation of more organic artwork with spontaneity than previously possible. Further development of the art traditions with new digital paint effects can also be achieved by tinkering with the simulated physics.*

1.6 Summary of Contributions

In the past six years, I have been developing a paint system prototype featuring novel methods of brush and paint simulation. This paint system (Figures 1.7 and 1.8), which I call *MoXi*, demonstrates the above thesis statement and has gain recognitions from both users and the industry. In 2006, *Adobe Systems Inc.* licensed our MoXi software and if things go well, the technology described in this thesis would be incorporated into

Adobe's product(s). In 2007, *Sony Electronics Inc.* also engaged into cooperation with us. For this, I ported the ink flow simulation part of MoXi to Sony's hardware. I believe my work has significant contributions towards the development of an advanced computer paint system and has also laid the foundation for others systems in the fields like computer-generated animation, computer-based education, Far-East font rendering.

The core of my thesis is on brush and water-based paint modeling. My brush model can deform more flexibly and accurately than previous work thus allowing more realistic marks to be made spontaneously and this is especially needed for Eastern calligraphy. It also represents a good balance of modeling efficiency in terms of accuracy and computation. My water-based paint simulation is based on a relatively new fluid-dynamics simulation method and is intrinsically suitable for running on modern graphics hardware. The resultant paint system using these two components delivers an organic painting experience not possible before in the digital realm.

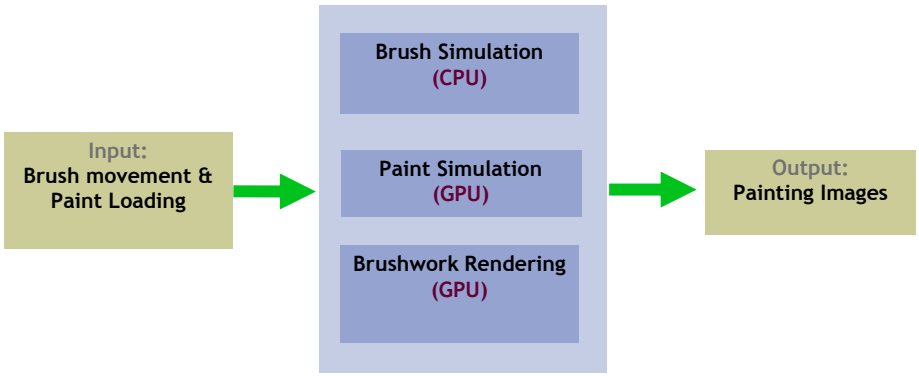


Figure 1.7: Block diagram of my paint system MoXi.

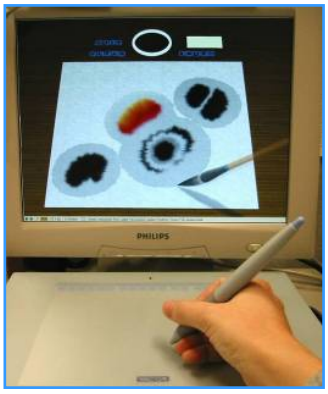


Figure 1.8: System setup with a graphics tablet as input device.

1.7 Thesis Organization

The rest of this dissertation is organized as follows. The next Chapter presents the technical details of my work on physically-based 3D brush modeling and philosophy behind its design. Chapter 3 describes my water-based paint modeling based on the Lattice Boltzmann method. Simulated art media include Eastern ink and Western watercolor. Reason for choosing the Lattice Boltzmann method and implementation details are also described. Chapter 4 discusses paint rendering. I present a method for enhancing the apparent resolution of the rendered artwork and discuss the use of the Kulbuka-Munk rendering model for water-based paint. Chapter 5 discusses user interface. This includes how the painting scene is presented to the user and a brief discussion on input devices. Finally, Chapter 6 concludes this dissertation with retrospection and discussion of further developments.

CHAPTER 2

BRUSH MODELING

Brushes are the single most important group of objects in an artist's studio. Having the right brushes is essential to the success of a painting. Often, the brush is considered as an extension of the artist's hand. Chinese ink painter Ning Yeh even says it is a waste of time if you use a low quality brush because you can never get the right stroke no matter how hard you try.

What drove me into paint tool modeling in the first place was my need for making Chinese calligraphy strokes on a computer. In the following exposition, I will focus on modeling a round Chinese brush first. The basic modeling technique was first presented in [Chu & Tai 2002]. The technique is also published in [Chu & Tai 2004] when speed optimization and brushes with multiple spines were added. Later in this Chapter, I would discuss how this could be extended to model flat brushes.

2.1 What is so special about Chinese Brushes?

The Chinese brushes may *look* the same as a Western watercolor round brush but actually they are of different designs. Before I continue, allow me to do a 'little' explanation of how and why Chinese brushes are different.

2.1.1 Brush Design

Rather than for just simple dabbing, the Chinese brushes are designed for making expressive lines [Silbergeld 1982; Barrass 2002]. A typical brush has layers of different types of animal hairs to give a good balance of absorbency and springiness. The tip of the brush forms a sharp point naturally when wetted (Figure 2.1 left) and is regarded as the part that gives the brush its spirit. One significant design feature is that a layer of shorter hairs called *mantie* is inside the brush (Figure 2.2). This creates some empty space serving as a reservoir and makes the root more resilient. When pressed lightly, the

body of the brush does not bend as much (Figure 2.1 right). This allows subtle control because movement at brush handle is well translated to the tip. This is good for calligraphy as delicate strokes can be made using only the tip, but wider strokes can also be made when pressed to the belly.

Figure 2.1: Comparison between typical Chinese (left) and Western watercolor (right) brushes. Left: Intrinsic shape. Right: When pressed lightly.

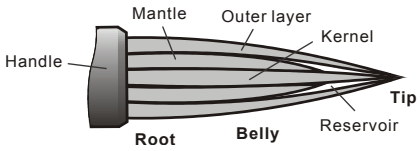


Figure 2.2: Anatomy of a typical Chinese brush.

As the ink on the brush is gradually consumed, the brush would break into smaller tufts. The brush tip can also be split deliberately to draw multiple lines or dots with a single stroke, or the tip can be nipped to form a flat end for painting things like a bamboo trunk. Chinese brushes are bigger then watercolor brushes and this also contributes to the variety in size and quality of possible marks one single brush can make. Chinese brushes are mostly general-purposed and rounded. There are also flat brushes used in Chinese art, but they are used less often and are usually for laying broad strokes only. Specialty brushes like those with long and stiff bristles for making exotic marks are also available. Worn brushes are often used for making scratchy texture.

2.1.2 Art and Usage

Traditionally in the West, the basic training for painting is *sketch drawing*, while, to the surprise of many, the corresponding training was, and basically still is, *calligraphy* in China. The former emphasizes on *depiction accuracy* while the latter *brush control*. Deft manipulation is required for effective use of a Chinese brush. This dexterity stems from

the needs in making beautiful strokes in Chinese calligraphy. In fact, Chinese painting (in the *spontaneous style,* that is) is also very much influenced by calligraphy. Chinese painting is characterized by the economy and the quality of the strokes. In making such strokes, rhythm is very important. Chinese painter Ning Yeh said *"the rhythm of a stroke is a happening, a serendipity."* Scholars often compare calligraphy with dancing and I myself could often *read* the strokes as if they are 2D records of the 3D movements of the brush.

In a typical technical book on Chinese painting, e.g. [Kwok 1981], readers would find demonstrations of making different dots and lines using the Chinese brush. However, the most convincing would be to watch the artists in action[2]. Here I just want to mention that one usage difference in Chinese and Western brushes is that the Chinese painters tend to hold the brush perpendicular to the painting surface and utilize only the tip part of the brush, while Western painters would use the side of the brush to lay down paint, mostly with the brush held slanted [Barrass 2002].

2.1.3 What's in a Name

We might also have a glimpse of how Chinese brushes are different when we look at the language used. Brushes used for calligraphy or painting are written as '筆' (Chinese pronunciation Romanized as 'bi', and Japanese 'fu-de'). For brushes used in sweeping or dabbing, as in house painting or cosmetic application, another term '刷子' (Romanized as 'shua-zi') is used. In English, both would be referred as 'brush'. There seems to be no direct single equivalent word in English for '筆' (the term 'paintbrush' refers to a brush for applying paint, but not necessary for making graphic). I myself would translate '筆' as 'graphic brush' to show its purpose for making graphic (others translated it as 'writing brush'). The development of '筆' from '刷子' was attributed to the need for expressive lining instrument by He [2001]. To put this in context, I would like to mention that Chinese people used different types of hair for giving different absorbency and springiness as early as the Han Dynasty (206 BC – 220 AD).

[2] Video samples can be found at http://www.orientalartsupply.com/products/brushes_combination.cfm

2.1.4 Concluding Remarks

In writing the above, I do no intend to suggest that Western brushes are inferior to Chinese brushes, or vise versa. Brushes from both East and West have evolved to wide range of instruments tailored for different painting purposes and brush makers from both sides certainly have all the skills to make the finest brush. I believe it is mainly the different media (oil, watercolor or ink) and the need (daily writing in ancient China) that cause the differences in brush qualities.

Nevertheless, I believe style and tools are slowly converging. Many artists have blended in Chinese techniques into Western watercolor paintings, and visa versa. Some watercolor brushes are now made in China and the manufacturers also advertise that many types of hairs go into a single brush just like the case for Chinese brushes. Thus, the distinction between Chinese and Western may be further blurred in the future. And, now digital tools probably would accelerate this process.

2.2 Related Work

People usually say we *paint* with a *brush* (which has flexible bristles or hairs) and *draw* with a *pencil* or *crayon* (which is solid). In many software tools, 'brush' is the single term used to refer to both wet and dry media. In the current review of previous work, I focus on painting with a brush as in watercolor or oil painting instead of drawing with some dry media. I will also limit the scope to painting on 2D rather than 3D surface.

2.2.1 Commercial Programs

In the commercial realm, the specific software application whose technology and development we should look at is 'paint program' (e.g. *Corel Painter*). But nowadays most 'image editors' (e.g. *Adobe Photoshop*), which allow editing of bitmap images, also include tools for 'painting' a stroke or 'drawing' a line or curve. Popular tools used by artists include *Adobe Photoshop, Corel Painter, AmbientDesign Artrage* and *Autodesk Sketchbook Pro*. The mark-making techniques used in these commercial programs are not published but as far as I can tell, all of them are 2D stamp or sweep based. These 2D mark-making techniques work very well for dry media like charcoal or pencil. *Artrage* can render impasto-style oil strokes quite realistically. However, since the brush model is basically a 1D sweep, the resulting strokes are still not organic

enough in my opinion. *Corel Painter* [Painter 2007] recently updated their brush simulation technology with what they call the 'RealBristle' engine in early 2007. However, after trying out, my feeling is that it is basically the same 2D mark-making technique, but runs faster and the strokes are rendered with more bristle-level details.

2.2.2 Academic Efforts

Earlier efforts in brushwork simulation focused on the rendering of strokes. Inspired by Japanese painting, Strassmann [1986] swept a one-dimensional texture to achieve varying shades within a stroke. These strokes look artificial because the natural spreading of the brush hairs is not adequately modeled. Hsu and Lee [1994] proposed the *skeletal stroke* technique, which deforms some pre-defined 2D strokes to produce remarkable results. However, for Chinese brushwork, which is very much textured, this technique requires storing a large sample of stroke textures to avoid appearing repetitive. The strokes also appear unrealistic whenever there is self-intersection or high curvature.

Later research efforts incorporated physical behaviors or physics theories into the brush models. Wong and Ip [2000] modeled a calligraphy brush as an inverted cone. The bulk of the cone has to penetrate the paper while stroking, with its footprint taken as the intersection of the cone with the paper plane. Since the brush tip is ignored while generating the strokes, this model fails to produce the *biased-tip* strokes (in which the tip travels along one side of a stroke rather than staying in the middle), which are commonly applied in painting and calligraphy [Kwok 1981]. Based on the theory of elasticity, Lee [1999] modeled a brush as a collection of rods. This model suffers from unnatural bending due to the assumed homogeneous elasticity used. Saito and Nakajima [1999] used a Bezier spine curve and a set of disks centered along the curve to model the brush. The model does not consider brush flattening and spreading and thus fails to generate realistic footprint. Compared to the cone model of Wong and Ip, the first model of Xu *et al.* [2002] has a more complex geometry and can split into smaller tufts; but the same problem of bulk penetration still leads to unrealistic brush footprints. In their second design, Xu *et al.* [2003] attempted to improve the brush dynamics by querying a motion database prepared from real brush motion data; however, the non-trivial data acquisition was not described. Girshick [2004] also embarked on the journey of brush

making. His model was heavily influenced by that of Xu et al [2003]. The dynamics was not done according to physics law but some rules devised by Girshick. His brushes tend to behave lesson realistically than those physically-based ones. After at least a few 3D brush models were proposed, Mi *et al.* [2002] chose to go back to 2D in their droplet brush model. They propose the 'droplet' brush model. The droplet footprint switches to different shapes by rules devised by them.

In contrast to Chinese brushes, there is much less research work focusing on modeling Western brushes. Baxter *et al.* [2001] modeled Western brushes as simple spring-mass systems, emphasizing the recreation of 'sight, touch, action and feel' of the painting process. No attention was paid to bristle spreading or splitting. Baxter later also implemented an energy-optimization-based brush model with anisotropic friction and plasticity [Baxter & Lin 2004]. In terms of modeling features, their new brush is quite similar to ours [Chu & Tai 2002; Chu & Tai 2004], but Baxter emphasized on making a full range of round, flat and fan brushes using both skeleton-surface and strip-based geometric representations. Adams *et al.* [2004] presented a 3D brush models based on those in [Baxter *et al.* 2001] but Adams' brush can split geometrically. Laerhoven and Reeth [2007] have also implemented an energy-optimization based brush model which is very similar to [Baxter & Lin 2004].

2.3 Approach Overview

My challenge was (and still is) to develop a model that can collectively simulate the hairs in real-time, and is flexible enough to produce the effects expected by artists. I believe that the deformation of the brush, which causes the ever changing footprints, plays a key role in producing convincing brushwork. Therefore, I have chosen a physically-based approach to model the brush dynamics (elaboration later in this section). My brush model consists of four components: *brush geometry*, *brush dynamics, ink loading,* and *ink depositing*. I first describe the model for one single tuft. The use of multiple tufts for effective modeling of wider brush spreading is described in Section 2.5.7.

2.3.1 Feature Comparison

Before presenting the technical details, it is useful to show visually the new features of my model over previous work. Figure 2.3 compares the different brush deformations and the corresponding footprints produced from our prototype system when the various modeling features are turned off. Figure 2.3(a) shows a fully functioning tuft being pressed against the virtual paper. Observe that the hairs split and spread laterally to produce a wider footprint. Figure 2.3(b) is obtained when the texture-based bristle splitting effect is turned off. If lateral spreading is further removed, we get the result in Figure 2.3(c), which is similar to results of previous physical models [Saito & Nakajima 1999, Baxter *et al.* 2001]. If brush-paper collision is not handled, the brush simply penetrates the paper as shown in Figure 2.3(d), and the footprint is a cross-section of the brush geometry, which resembles the effect of other previous models [Wong & Ip 2000; Xu *et al.* 2002].

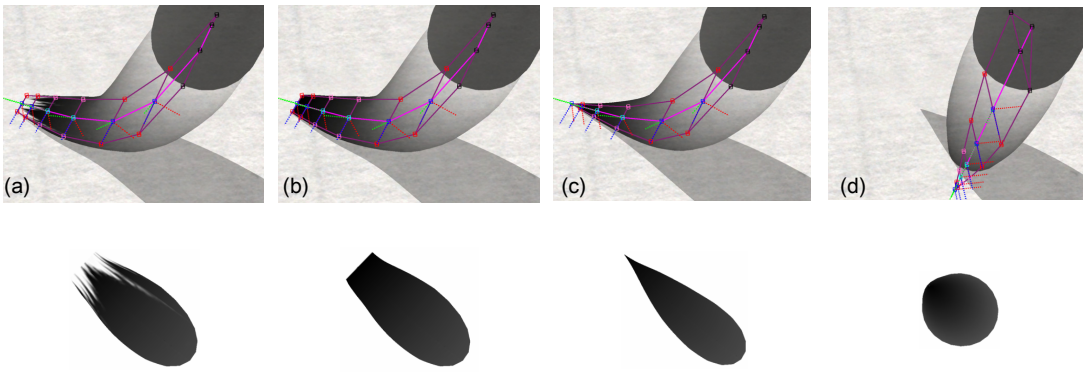


Figure 2.3: A comparison of tuft deformation effects and corresponding footprints. (a) My brush model in full gear. (b) Without split map. (c) Without split map and lateral spreading. (d) Ignoring brush-paper collision.

2.3.2 Why 3D and Physically-based?

With 2D mark-making methods, it is certainly possible to draw *any shape,* either by dragging control points of curves or by painting with variable dab size. Simulating brush dynamics using a 3D brush model certainly requires more computational power, so why would one use a 3D brush model rather than a 2D one?

In real-life brushwork, the formation of a stroke is largely governed by the constantly changing brush footprints while the brush is manipulated. A good 3D brush model should generate realistic brush footprint automatically from the brush posture. Without such a model, it is difficult for an artist to produce a stroke that looks as if painted in a *bouncing rhythm* with an elastic brush. Rhythmic vitality, which is the essence of Chinese art, is lost in the process of adjusting control points or parameters. This is analogous to the need for motion capture to provide *lively* character motions in computer-generated animation. When executing spontaneous strokes with a real brush, it is the elasticity of the brush that completes the stroke vibrancy; that is, the response of the flexible brush itself to the artist's motion also plays an important part in producing the final appearance. This is essentially why strokes made with Chinese brushes can be so eloquent.

Modeling the physics of the brush dynamics offers the possibility to extend the expressive power of real elastic brushes to the digital domain. Moreover, a 3D physical model allows visual feedback of the brush shape. Coupled with appropriate input device, such a brush model makes a painting system intuitive to artists. In view of these benefits, I pursue the design of a physically-based 3D brush model.

2.4 Brush Geometry

I represent the geometry in two layers, namely, the *skeleton* and the *surface*. Note that similar skeleton-based models are very common in other areas in computer graphics like character animation [Chawisk *et al.* 1989] and hair modeling [Plante *et al.* 2001]. In my opinion, modeling hundreds or even thousands of hairs geometrically is not really necessary. From an artist's perspective, what really matter are the natural shape of the marks and that the output is amenable to high resolution rendering.

2.4.1 Brush Skeleton

The skeleton consists of a *spine* and some *lateral nodes* (Figure 2.4). The spine is responsible for the general bending of the whole tuft, while the lateral nodes model the lateral deformation and spreading of the tuft. The spine is represented as a connected sequence of line segments, with progressively shorter segments towards the tip. Higher resolution is dedicated towards the tip because the tip is softer and that usually only the

tip and the belly are used to paint. Each joint between two segments has two DOF's in the latitude ϕ_i and the longitude θ_i of a spherical coordinate system (Figure 2.5), referred to as the *bend*- and the *turn*-angles at that joint respectively.

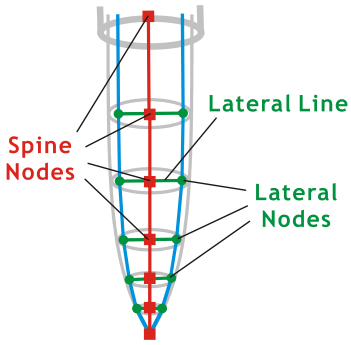


Figure 2.4: Geometric model of a brush tuft.

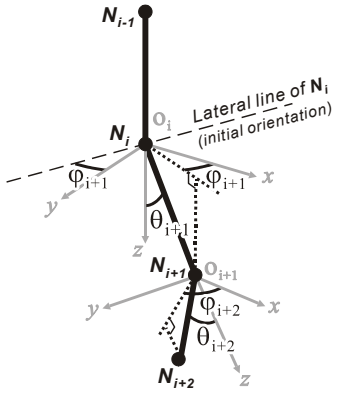


Figure 2.5: Notations for our geometric model.

The position of a spine node $\boldsymbol{N_i}$ is denoted as O_i , $i = 0,\ldots,n$, with O_0 being the position of the brush root node. Each spine node is attached with two lateral nodes, which are constrained to move along the *lateral line* of the spine node. The lateral line of $\boldsymbol{N_i}$ passes through O_i and is initially perpendicular to the two adjacent spine segments. It has one rotational DOF on the *joint-bisecting plane* of the spine node, which is defined to be the plane passing through O_i and bisecting the angle between the two adjacent spine segments.

The two lateral nodes attached to a spine node represent two groups of hair on both sides of the spine. We observe that this design can effectively capture the essence of tuft deformation for the following reasons. Since the brush interacts with only a planar painting surface, the brush footprint is largely determined by tuft flattening, controlled by the bending and lateral drag. With the lateral lines having one DOF, the non-penetration constraints in our dynamics model tend to keep the lateral lines of those spine nodes that touch the paper to be parallel to the paper. Consequently, when the brush is pressed against the paper, the loci of the lateral nodes would lie on the painting surface, and thus effectively model horizontal deformation and the lateral spreading of the bristles.

2.4.2 Brush Surface

I represent the brush surface as a swept surface defined by the spine and a varying elliptic cross-section. When the brushes are moistened and unbent, the cross-sections are circles along the entire spine. I pre-define these *initial tuft radii* for various types of brushes. In general, the cross-section is composed of two half ellipses, with possibly different major radii, but a common minor radius (Figure 2.6). This simple representation is computationally efficient and does not differ much from real brush tufts. The cross-section Ω_i at spine node $\boldsymbol{N_i}$ lies on the joint-bisecting plane, and its major axis coincides with the lateral line of $\boldsymbol{N_i}$. Let a and b be the distances between $\boldsymbol{N_i}$ and its lateral nodes (Figure 2.6). The major radii for Ω_i are taken as $(a + r_{lat})$ and $(b + r_{lat})$, where r_{lat} is the *effective radii* (described in Section 2.5.4) for the two lateral nodes. The common minor radius, c, is then computed by the conservation of area:

$$c = \frac{r^2}{\frac{1}{2}(a+b) + r_{lat}}$$

where r is the *initial tuft radius* at $\boldsymbol{N_i}$.

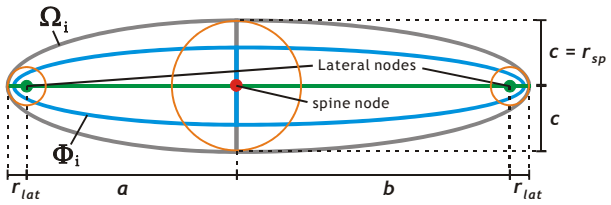


Figure 2.6: Tuft cross-section.

2.4.3 Fine Hair Effect

To obtain footprints with *hair-level* brush splitting effect while using a single tuft, we define an alpha map, named the *split map,* to make part of the tuft surface transparent. The split map can be either prepared from a scanned image of real bristles or generated by patching some mask primitives on the fly (Figure 2.7). We observe that a *static* split map applied on a single tuft, together with brush flattening, is sufficient for simulating convincing bristle splitting as long as the brush is not pressed too hard (Figure 2.3(a)). For simulating the effect of wider spreading, we propose multiple tufts that work in a

hierarchy (see Section 2.5.7), with split maps still applied for fine hair effect. Split maps increase the modeling efficiency dramatically by being an inexpensive feature supported by graphics hardware and producing ink streaks that look very natural.

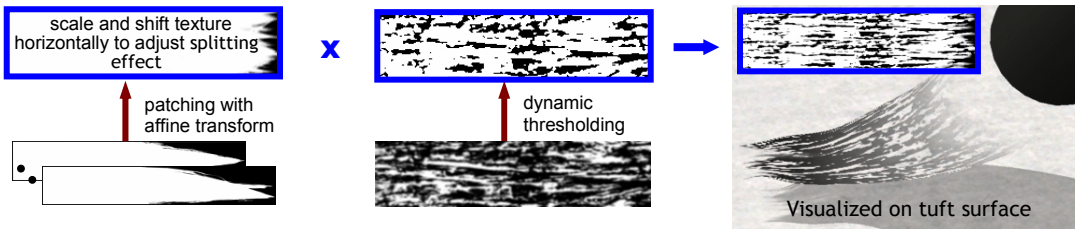


Figure 2.7: Texture-based ink depositing effects

2.5 Brush Dynamics

In the past two decades, graphics researchers have adopted physics simulation to model soft objects in order to produce realistic animation. In particular, a lot of effort has been put into cloth simulation for animating clothed characters in computer animation. Because there has been a much larger body of research on cloth simulation than brush simulation, it is helpful to examine some of the previous work on cloth simulation techniques.

A good survey of cloth simulation techniques can be found for example in [Ng & Grimsdale 1996]. Today, one popular way to simulate cloth dynamics is to model it as a set of particles connected with springs. Paramount motion equations (using Newton's second law of motion) are then set up and solved for to determine the positions of the particles. Another class of method is energy minimization. Feynman [1986] was probably the first to simulate cloth by energy minimization in the computer graphics community. He claimed that energy of a piece of cloth is locally minimized when it is in a stationary state, and that this is strictly true *only* in the absence of friction, viscosity and other factors that can convert motion into heat.

Cloth strongly resists stretching motions while being comparatively permissive in allowing bending or shearing motions. This results in a "stiff" underlying differential equation of motion [Baraff & Witkin 1998]. Explicit methods are ill-suited to solving stiff equations because they require many small steps to stably advance the simulation

forward in time. In practice, the computational cost of an explicit method greatly limits the realizable resolution of the cloth.

2.5.1 Why Energy Minimization?

We refer simulation techniques that solve motion equations in vector form as *vectorial methods* [Tabarrok & Rimrott 1994]. Cloth is easy to fold but highly resistance to stretching. Keeping particles from excessive stretching requires large spring coefficients. This again induces high acceleration in the motion equations. Solving for these equations means integrating them with tiny time steps – steps that are much smaller than the frame during of a typical animation. Due to the 'stiff' nature of the differential equations cloth [Baraff and Witkin 1998], it is not easy to simulate a large system in real-time even with today's computational resources. In fact, stiffness is so nasty that Baraff and Witkin [1999] would even recommend avoiding stiff equations whenever one can. But how can one avoid stiff different equations? For soft object simulation when inertia is important, it is not obvious how. Energy minimization has been used [Breen *et al.* 1994], but it was only to predict the drape of woven cloth rather than bulk inertial motion. In rigid-body dynamics simulation, it is well-known that traditional vectorial techniques have problems in handling stacks of objects or otherwise 'crowded' scenes. The problems are caused by the frequent collision response involved and this is analogous to 'stiff' situation in soft-body simulation. Milenkovic and Schmidl [2001] proposed the 'optimization-based animation' technique to alleviate the problems in rigid-body dynamics and showed promising performance and stability. However, they also state that their simulation is physically approximate, and is good for visually *plausible* animations. Milenkovic and Schmidl also mentioned that another benefit of using optimization is that techniques are already well developed by mathematicians and computer graphics researchers can simply harvest the fruits.

Turning back to brush modeling, Saito and Nakajima [1999] were the first to model brush dynamics with energy minimization. However, they did not explain why energy minimization would work. Given the stiff nature of the dynamic system, it is difficult to produce a tractable real-time system for the brushes by solving second-order differential equations derived from Newton's second law. Baxter *et al.* [2001] cope with the

instability difficulty by adding large damping and employing an approximated implicit integrator [Desbrun *et al.* 1999], which is reported to be more stable and faster, but much less accurate, than the large step integrator presented in [Baraff & Atkin 1998]. However, the approximated integrator has the drawback of having to model the brush internal forces using only stretch springs rather than bend springs, which model non-stretchable bristles more naturally. Baxter later [Baxter & Lin 2004] also implemented a brush model based on energy-minimization. It is interesting to note that Baxter did not include his first work [Baxter *et al.* 2001] on brush simulation, which is based on vectorial method, but only his later brush model [Baxter & Lin 2004] into his PhD thesis [Baxter 2004].

In Physics nomenclature, energy minimization belongs to *variational* methods [Tabarrok & Rimrott 1994]. While vectorial methods deal with the acceleration and velocity of objects at *specific* time moment, variational methods talks about changes in properties of objects over an *arbitrary* amount of time. Time-stepping integration is effectively avoided in the latter. In view of the ability to avoid time-stepping integration altogether and the easy of implementation, I have chosen the variational method of energy minimization.

2.5.2 Why Energy Minimization Works

Baxter and Lin [2004] used the *principle of virtual work* to explain why energy minimization can be used to predict the brush motion. This, however, is open to discussion. The principle of virtual work (PVW) is a classic principle in solid mechanics which states:

> *the total virtual work done by all the forces acting on a system in static equilibrium is zero for a set of infinitesimal virtual displacements from equilibrium.*

The infinitesimal displacements are called virtual because they need not be obtained by a displacement that actually occurs in the system. The virtual work is the work done by the virtual displacements, which can be arbitrary, provided they are consistent with the constraints of the system.

Typically, PVW is used to solve static problems [Meriam & Kraige 1992]. Once the system goes through some *real* motion the problem becomes a dynamics one. Is PVW still applicable? Does it work when dissipative force is involved as in our case of brush simulation? Robotic researchers Peshkin and Sanderson [1988] stated that PVW makes no prediction of motion. Furthermore, the principle is not valid for systems involving friction [Peshkin & Sanderson 1988; Goldstein 1980].

However, Peshkin and Sanderson [1988] claimed that energy minimization can be used to predicted the motion of mechanical systems in which mass or acceleration are sufficiently small that the inertial term ma in $F = ma$ is negligible compared to dissipative forces. Such systems are refereed to as *quasi-static*. They proposed the *principle of minimum power*, which states:

> *A quasi-static system chooses that motion, from among all motions satisfying the constraints, which minimizes the instantaneous power.*

They proved that their proposed principle is correct in the special case that Coulomb friction is the only dissipative or velocity-dependent force acting in the system. In our case, a brush system can be regarded as a *quasi-static* defined above because a plastoelastic brush moves as if it has no inertial. This means if we formulate the friction between my brush and paper as Coulomb friction, which should be naturally so, and have no other forces acting on the brush, then, viewing from this *quasi-state* angle, our use of energy minimization should be valid.

In mechanical engineering literature, researchers also proposed a new variational principle regarding the prediction of motion of a solid object in the presence of friction in a time-discretized setting [Kane *et al.* 1999; Pandolfi *et al.* 2002]. They introduce the concept of *incremental potential energy*, which comprises terms accounting for inertia, strain energy, contact, friction and external forcing, which are just what I need to model in my brush simulation. The variational principle proposed is of *'a radically different nature than the variational principle of classical mechanics'* [Radovitzky & Ortiz 1999; Pandolfi *et al.* 2002] and therefore is different from PVW. They claim that using this principle, the minimization of the above *incremental potential energy* delivers a solution

consistent with the equations of motion of solids. Our formulation of energy (Section 2.5.4) fits well in framework of Pandolfi *et al.* and so I believe the *minimum principle of incremental potential energy* can be the theoretical basis for our energy minimization approach.

2.5.3 Energy Minimization Problem

We formulate the brush dynamics as a series of static constrained minimization problems. The objective functions are the current incremental potential energy of the dynamical system, and the constraints are that the brush geometry must be above the paper. The set of variables Ψ in the minimization comprises all the joint angles and the stretches of the lateral nodes. Our incremental potential energy has three components: *strain energy, internal frictional energy* and *external frictional energy*. We do not include a component accounting for inertia because real brushes reach equilibrium almost instantaneously as they are being pressed against or lifted from the paper. The *strain energy* accounts for the potential energy stored in the brush due to deformation from its unstrained state. The *internal frictional energy* refers to the work done against the internal friction between water and bristle molecules as the tuft is deformed. The *external frictional energy* is the energy lost to friction between the brush and the paper surface.

We derive the strain energy of the brush by setting up a spring system. *Bend* and *twist* springs are imposed on the bend- and the turn-angles at the spine nodes respectively to model the general deformation of the tuft. *Stretch* springs are placed between the spine nodes and their lateral nodes to model lateral deformation. Finally, we also put some bend springs between consecutive nodes to account for the stiffness of the bristles on the sides of the brush. The details of formulating these spring energies as well as the frictional energies are presented in Section 2.5.4. We observe that such behavior functions suffice for producing plausible brush movements.

Inequality constraints are introduced in the minimization to account for the normal reaction force of the paper on the brush. With the paper plane being $y = 0$, we can express the constraints as

$$g_i(\mathbf{x}) = y_i > 0$$

where y_i is the lowest point of an ellipsoid generated by rotating Ω_i about its major axis. However, we allow part of the brush surface to penetrate the paper in order to generate the brush footprint (see Section 2.7); the penetration also makes the brush appear to have a flat bottom when pressed against the paper. Hence, for collision detection, we use a shrunk version of Ω_i, denoted by Φ_i (Figure 2.6).

We chose to solve the energy minimization problems by local Sequential Quadratic Programming (SQP) due to its fast convergence (see Section 2.5.8 for further speedups). There are a few points we need to take care of when applying gradient-based optimization techniques like SQP. While a brush is dragged on paper, minimizing the external frictional energy means minimizing the number of nodes that are in contact with the paper, thus causing the brush to be lifted. To avoid this unphysical lifting, we forbid the change of the touching state of the brush nodes during the search for the minimum; that is, for each time step, the touching state evaluated on the first iteration is used.

In general, we should also try to avoid discontinuity in the energy functions when formulating them so that the minimization would converge more easily.

2.5.4 Energy Functions

Strain Energy. The energy stored in a *bend* or *twist* spring is in the form

$$E(\Psi) = \kappa_i |\theta_i|^m$$

where κ_i is the spring coefficient. Our current implementation uses $m = 2$ or 3. In general, spring coefficients are determined empirically, with the assumption that brush root is stiffer than the tip. We omit a twist spring at the root so that the tuft has no preference of bending in a particular direction.

We take the distance of a lateral node from its spine node as $r_i + \delta_i^2$, where r_i is the initial tuft radius of the spine node plus the effective radius (described below) of the lateral node, and δ_i is the variable in the minimization that models the stretch DOF. The energy function for a *stretch* spring attached to a lateral node is then in the form

$$E(\Psi) = \kappa_i \left| \delta_i^{\,2} - S(\theta_i, p_i) \right|^m$$

p_i is the proportion of the associated tuft segment underneath the paper. S is a heuristic function of both θ_i and p_i that increases the rest length of the spring as the brush is pressed against the paper.

Internal Frictional Energy. This is derived by applying the same form of energy functions used for the bend energy on the *change* of the joint angles and lateral stretches since the last frame.

External Frictional Energy. For every node contacting the paper, we calculate its frictional energy as

$$E(\Psi) = \mu \cdot R_i \cdot x_i$$

where x_i is the dragged distance, R_i is the normal reaction force, and μ is the coefficient of friction. A node is regarded as contacting if its height minus its *effective radius* is below the paper. We denote the effective radii of a spine node and its lateral nodes by r_{sp}, and r_{lat} respectively (Figure 2.6). r_{sp} is set to be equal to the minor radius c of Ω_i from the previous time frame while r_{lat} is taken as a fraction of r_{sp}.

Real wet bristles are slightly adhered to the paper due to moisture. In our simulation, we also want a wet brush tip to experience some friction even when its normal force is zero. So, for each wet node, we also add to its R_i a small value proportional to its p_i.

Since the hair on a brush is generally aligned, the brush would experience a larger resistance when dragged sideway. To provide control of this anisotropic behavior, we also modulate μ with the direction of the spine segment.

2.5.5 Plasticity

When a wet brush is pressed and deformed, energy is lost to the friction between the bristles and the water within them. On releasing the brush, the restoring spring force has to overcome the resistance of this friction to revert to its original shape. Failing to do so makes the brush appear plastic.

When incorporating plasticity, my first consideration is to add a frictional energy term to the total energy functional. This frictional energy is due to the shifting of the wet hairs along themselves when they are bent. This shifting distance can be approximated by the change in bend angles $\Delta\beta_i$ at the spine nodes. I denote this added term by E_{plas} :

$$E_{plas}\,() = \Sigma\,(\,k_i\,|\Delta\beta_i|)$$

where k_i are weights. However, the function $E_{plas}()$ that takes the absolute value of $\Delta\beta_i$ is not smooth at $\Delta\beta_i = 0$, making it hard to find solution for the SQP problem. In practice, this results in the brush dynamics behaving a bit jerky. It is possible to square (or raise it to a power > 1) instead of taking absolute value of $\Delta\beta_i$ so that E_{plas} () is smooth, but then the result would be that the brush gradually straightens when the brush is not pressed (assuming energy minimization is always being applied). The reason is that the local minima of objective function is now blunted (as we have deliberately made it so) and that the local minima's position is 'reset' every frame: a small change in β_i gives almost the same $E_{plas}()$, but smaller β_i gives smaller bend energy. Thus, SQP tends to move the brush towards configurations with smaller β_i while $E_{plas}()$ is smaller enough. I attempted to stop this undesirable behavior by applying energy minimization only when the brush touches the paper, but then in practice if we lift the brush abruptly, the brush keeps in its bent position unnaturally.

I finally devised another method to model this plasticity. The intuitive idea is to make the brush have a high tendency to remain in the skeleton configuration of the last time frame. To achieve this, we simply shift the spring energy functions so that the lowest energy is at the position corresponding to the last skeleton configuration. To avoid having the brush appears overly deformed, we also clamp the shifted distance against a user-adjustable plasticity parameter α. Suppose the original bending energy function is $E(\theta) = k|\theta|^m$. If θ' is the bend-angle from the previous time frame, the new bending energy is then

$$E(\theta) = k\left|\theta - \rho\right|^m$$

where $\rho = \text{clamp}(\theta', -\alpha, \alpha)$. This simple but effective method significantly improves the realism of the brush, producing the plastic behavior expected by users of real brushes. Brush plasticity does affect the rhythmic movement made by artists and thus indirectly determines the ink traces.

In real-life Chinese painting, re-shaping the brush tip to a sharp point is often necessary before drawing a new stroke. If desired, the plasticity can be set to zero so that tip re-shaping is eliminated altogether. In our implementation, the brush can be reverted to its original shape at any time by a user command.

2.5.6 Pore Resistance

Most types of painting paper are full of pores. When the tip of a slanted brush is brought into contact with the paper, the pores act like a fence impeding sliding. If the brush is then pushed towards the direction pointed to by the tip, these pores will continue to exert large resistance (Figure 2.9), producing the rough texture seen in *pushed strokes* [Kwo81] (Figures 2.17(b)) employed in many paintings. To simulate such brush behavior and painting effect, we also model paper pore resistance. Our approach is to add an extra constraint in the minimization problem in the form of a *moving blocking plane* when the tip touches the paper. The blocking plane is vertical and normal to the projected spine segment of the brush tip onto the paper surface. To adjust the blocking effect, we increase the lead distance L between the plane and the brush tip as follows:

$$L = at + bu + c$$

t is defined as $\max(|\beta| - \gamma, 0)$, where β is the angle the spine tip segment makes with the paper normal, and γ is the *critical tilt angle* within which the tip gets trapped by the pores. u is the pressure experienced by the tip. a, b, and c are user-adjustable parameters.

A soft brush may be too plastic to spring back to form a tip needed for making a blade-like stroke ending (Figure 2.17(d)). Calligraphers cope with this by using pore resistance to straighten the brush. Mimicking pore resistance in our simulation allows the reproduction of this kind of deformation, which is expected by experienced artists.

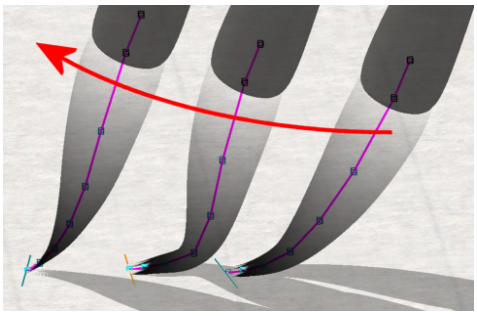


Figure 2.8: Pore resistance deforming the brush. The red arrow indicates the brush motion. The blocking constraint is indicated by a cyan line which turns orange when the constraint is active.

2.5.7 Multiple Spines

A thoroughly moisten brush forms a single tuft. The interaction of the brush with the paper surface and the reduced moisture, however, cause the brush tuft to break into smaller tufts during the course of painting. When splitting begins, certain types of brushes tend to have some 'unruly' bristles at the outer layer (Section 2.1.1) sprouted outward, forming thin tufts acting like satellites. Other types of brushes with even bristle distribution tend to split into tufts of even sizes. All brushes finally become bushy when they are really dry. When a brush splits, multiple lines are drawn simultaneously, and the small tufts can go through slightly different paths producing subtle ink streaks (Figure 2.17(b)(h)).

To better simulate the splitting of real brushes, we introduce multiple tufts arranged in a hierarchy so that geometric split can be realized (Figure 2.9). A child tuft T_j is associated with a *parent node*, which is a spine node N_{i_j} of its parent tuft. The root of the child tuft T_j lies on the joint-bisecting plane of N_{i_j} and is confined within the elliptic cross-section Ω_{i_j}. Its exact location is denoted by the lateral displacement $\boldsymbol{r_j} = (r_{xj}, r_{yj})$ from Ω_{i_j}, defined along and normalized by the lengths of the major and minor radii of Ω_{i_j}. There are a number of factors that causes sprouting to occur: displacement of lateral nodes, pore resistance, bending or drying of brush. We employ some simple heuristics on the friction, the bend-angles and the wetness of the parent tuft to decide

when to sprout child tufts and what their parent nodes are. The lateral displacements are determined according to the factors that cause the sprouting:

1. Due to displacement of a lateral node: $\boldsymbol{r}_j$ lies in the vicinity of that lateral node.
2. Due to pore resistance: $\boldsymbol{r}_j$ is lies in vicinity of the minor axis of Ω_{i_j}.
3. Due to bending or drying up: $\boldsymbol{r}_j$ is randomized within Ω_{i_j}.

For efficiency, the parent and its child tufts are allowed to intersect with each other. We do not enforce conservation of total tuft volume since, in reality, a split brush may become bushy. For modeling thin child tufts sprouted from the outer hair layer, we let the child tufts have default flattened shapes; for modeling brushes that split into even-size tufts, we replace the main tuft by a number of smaller tufts without any parent-child relationship like [Xu *et al.* 2002].

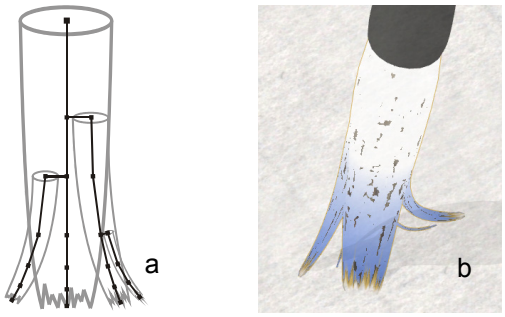


Figure 2.9: (a) Multiple tufts in a hierarchy. (b) A brush with four child-tufts spouted.

A new tuft has the same spine structure as its parent from the tip up to its parent node, and is initially assigned the same bend- and turn-angles. Once a new tuft is created, its lateral displacement is fixed, but the tuft can split further upwards – the parent node may become N_{i_j-1}. The child tuft then grows by one segment at its root. A split brush can be reverted to a single tuft by a simple reset.

Currently, the skeletal configuration of a child tuft is solved for via a separate energy minimization problem after the configuration of its parent is found. The computational demand of simulating numerous child tufts can be reduced by eliminating

lateral nodes of thin tufts. These thin tufts can still exhibit flattening, which is controlled by a function of only the bend-angles of its spine. We observe that a small number of tufts (at most five) suffice for most painting purposes; larger number of tufts are needed only when applying special painting effects like rubbing with a bushy brush to produce rough texture (like Figure 2.13(e)). In such cases, it may not be worthwhile striving to simulate dozens of tufts with accurate physics because the essence of such strokes lies on the controlled randomness rather than the brush elasticity.

2.5.8 Real-Time Simulation

There are a number of ways to speed up the simulation. It is well known that SQP converges quickly if the initial estimate is close to the solution. In this application, the brush state from the last time frame serves as a very good initial estimate. For the very first time step when an unbent brush first touches the paper, to get a good initial estimate for the turn-angle at the brush root, ϕ_0 , we initialize it such that the lateral lines are horizontal. If the brush is held exactly vertical when it touches the paper (which is rare), we simply set ϕ_0 to zero since there is no preference on this angle in this case.

The speed up largely depends on the correlation of successive minimization problems in our simulation. The SQP method has many variations; our current energy minimizer is based on a common implementation that iteratively updates a Hessian matrix L_{xx}, from which the search direction for the solution is derived. We modify the basic algorithm to use the resulting L_{xx} from the previous time step, instead of the identity matrix, as the initial L_{xx}. Exploiting the similarity of these matrices reduces the number of iterations needed significantly by up to 50%. This speedup over our earlier implementation [Chu & Tai 2002], together with improved code efficiency, has made multiple tufts practical in our current prototype.

We limit the number of iterations q in each search process to a maximum value q_{max} to maintain a reasonable frame rate. In our current prototype, we set q_{max} to about 20 iterations for each tuft of 6 segments. If the prescribed accuracy is not reached when q exceeds q_{max} in the current frame, the search process is effectively continued in the next time step since the Hessian L_{xx} is passed on. This is manifested when a tuft moves slowly towards its final state for several frames after the brush manipulation has

stopped. Table 2.1 compares the performance of our painting system with various dynamics simulation settings.

# 6-seg. tufts	q_{max}	Continuous Hessian update: No		Continuous Hessian update: Yes	
		Av. q	Fr./Sec.	Av. q	Fr./Sec.
1	20	14.6	60.5	7.4	62.6
3	20	14.6	30.4	7.4	44.6
3	25	15.4	28.8	8.2	43.3
3	30	16.7	28.0	9.7	42.6
5	20	14.6	18.9	7.4	28.5

Table 2.1. Overall frame-rates with various dynamics simulation settings. Experiment was performed on a 2.26G Pentium-4 PC with an NVIDIA GeForce 4 graphics card.

Care should also be taken in order not to waste computation on excessive accuracy. Unlike scientific applications, our brush simulation does not need to be very accurate for the brush to work well. Single-word precision is sufficient and empirically determined threshold values (e.g., convergence criteria) are adopted in the SQP.

2.5.9 Discussion on Skeleton Mechanism

The Euler angles are one way to describe the orientation of a rigid body in 3-dimensional Euclidean space. I have chosen this representation for defining tuft skeleton mechanism because they are intuitive to visualize and implement. As noted in [Baxter 2004], formally our bend and turn angles can be referred as the ZX angles in the Euler ZXZ angle set. One benefit of using ZX instead XY angles is that we get the bend angle without extra calculation. Baxter [2004] pointed out that using the ZX angles has the deficiency of having singularity at zero angles, which is equivalent to the brush at an unbent state. However, in practice, this could be easily avoided by setting a small angle when the brush a straightened.

One possible improvement on the dynamics simulation efficiency is to use an adaptive skeleton system in which at every time frame the joint mechanism is restructured to have the two degree of freedom rotations as a bend about a horizontal axis and bend about a vertical axis. Brush motions are mostly of two kinds: *press and lift*, and *sideway drag*. In gradient-based optimization, the search direction is along the dE/dx_i, where $E(x_1,.., x_n)$ is the objection function, and x_i's are the free variables. I expect

restructuring the joint mechanism is the above way would increase the chance that dE/dx_i are along the steepest slope of $E()$, thus speeding up the convergence. Further investigation on such adaptive joint mechanism or other rotation representation is left as future work.

2.6 Paint Loading

In real-life Chinese color painting, usually ink, water and colors are mixed right within the brush, by loading different colors up to different lengths of the brush followed by gentle stroking to blend the colors. Allowing color loading and blending using similar brush motions in a virtual brush system would be the most intuitive. In fact, this has already been done in [Baxter *et al.* 2001]. However, our current system employs a loading method which allows more precise control. Users define a color gradient by assigning colors ramps along the gradient, which is then mapped to the brush surface axially. Such interface is common in existing illustration software. We intend to implement the intuitive color loading interface as an option, together with the simulation of ink diffusion within the brush, in the future.

2.7 Paint Deposition

Paint depositing is defined as the problem of determining which parts of the paper is receiving ink and how ink is transferred from the brush to the paper. The first step in ink depositing is to determine the current brush footprint. Similar to [Baxter *et al.* 2001], we allow part of the brush surface to intersect with the paper plane (collision detection is discussed in Section 2.5.3) and consider the orthogonal projection of the penetrating portion onto the paper plane as the footprint. The footprint is obtained by rendering the brush surface clipped by the paper quadrilateral. The inkwork, which is stored as a texture for the paper, is updated with the ink values of the footprint. Next, we can either subtract the ink values from the brush or simply maintain the ink level to allow continuous painting without reloading. To apply transparent colors on top of existing colors, alpha blending is used currently (when paint flow simulation described in Chapter 3 is turned off). The generation of footprint and the blending of the deposited ink are all done on the GPU using OpenGL, thus minimizing data transfer and leaving more CPU horsepower for the physics simulation.

To produce more realistic ink traces, we model three ink depositing effects: *dry brush*, *soaked brush*, and *grain texture*.

2.7.1 Dry Brush

Bristles can be regarded as forming a height-field on the brush surface. When a brush is rather dry, only the peaks are depositing ink. In addition to the split map, we define another alpha map called the *dry map* for the tuft surface. The dry map is dynamically generated by thresholding a pre-defined gray-scale image (Figure 2.7), mimicking the gradual drying up of the brush. This gray-scale image is prepared from real dry-brush prints to give a natural feel. The threshold varies across the whole map reflecting the moisture level and the pressure at the brush nodes. Using the *multi-texturing* function of OpenGL, the final alpha texture applied onto the tuft surface is the split map modulated by the dry map.

2.7.2 Soaked brush

The tip of a soaked brush expands a little due to excess moisture; thus, ink streaks do not appear even if the tip splits slightly. To mimic this effect, we simply expand the tip geometry slightly and dilate the opaque regions on the split map. As the excess moisture is drained, the tip geometry shrinks back and the opaque regions on the split map contract back to their original shapes. In our implementation, the dilation and contraction are effectively achieved by scaling and shifting the split map down the length of the spine without dynamic texture generation.

2.7.3 Grain Texture

Paper grain texture emerges only when the brush is rather dry (Figure 2.13(e) and (j)). We use a gray-scale image to represent the paper height-field. The threshold is set according to the current wetness and the pressure of the brush to produce an alpha mask to be applied on the resulting brush footprint.

2.8 Flat Brushes

Western flat brushes are chisel shaped brushes (Figure 2.10 Left) that first became popular among Impressionist painters of the late 19th century [Handprint FlatBrush 2007]. They are good for laying down large areas of even color and creating shapes

(usually with one or more straight edges) less convenient to render with a round brush. Eastern 'ha-ke' (Romanization of the Japanese term '刷毛', which means '刷子' in Chinese) flat brushes (Figure 2.10 Right) are made of goat hairs which are softer and more absorbent. These are good for washes, and sometimes ink painters also use it for color gradations.

Figure 2.10: Left: Real flat brushes. Right: Natural splitting after moisture is used up.

Having observed the reality, I have a few design criteria in my flat brush model:

- The brush largely moves as a whole deformable object
- Occasional splitting should occur

To model a flat brush, I use multiple tufts arranged in a row, each represents one slice of brush's breadth (Figure 2.11). There are a few modifications I have to make in order to make the brush behave more realistically. First, I have to make the cross-section of the brush more *squared* instead of purely elliptic. This was done simply by changing formulae for generating the brush surface to: $x = a \cdot \cos^2(t)$ instead of $x = a \cdot \cos(t)$, where x is the x-position (in the local frame of the spine node) of the brush surface mesh node, t is a parameter for generating the cross-section, and a is minor radius of the brush cross-section. Note that if the y-position is also squared, the resulting shape is one form of the 'super-ellipse' [Mathworld Superellipse 2007]. I chose to only square the x-position because it is cost-effective and that the resulting texture coordinates are less distorted. Another cheap way is the have the middle part of the brush just flat in thickness and the two sides of the brush elliptic. Care should also be taken to make sure the texture coordinates are not too much distorted.

Another aspect that needs to be addressed is the coherence of the separate tufts. If flat tufts are naively set up so that they have nothing to do any other tufts, they would appear too incoherent and disjointed (Figure 2.11 middle). Note that in modeling round brushes, since the tips of smaller tufts are more pointed, the brush as a whole would not appear awkward. The way I deal with this is as follows. From my experimentation, it is not necessary to model buckling when a flat tuft is bent, because unlike a round tuft, the bristles of a short flat tuft do not push each other to the sides much. This allows me to have the spine having no lateral nodes. I then override the local *xyz*-frame (Figure 2.5) at each spine node so that the *y*-axis is along the vector $O_i^{t-1}\ O_i^{t+1}$, in which O_i^t denotes the position of the *i*-th spine node of the *t*-th tuft. If the *t*-th tuft has neighbor on only one side, the *y*-axis will then be along the vector $O_i^t\ O_i^k$, where O_i^k is the position of the *i*-th spine node of the neighboring tuft. This new *y*-axis makes the resultant brush surface to appear coherent with those of other tufts. With the above measures in place, the whole brush has the desired behavior of moving largely as a whole while occasional splitting still occurs as *each* spine controls *its own* surface mesh.

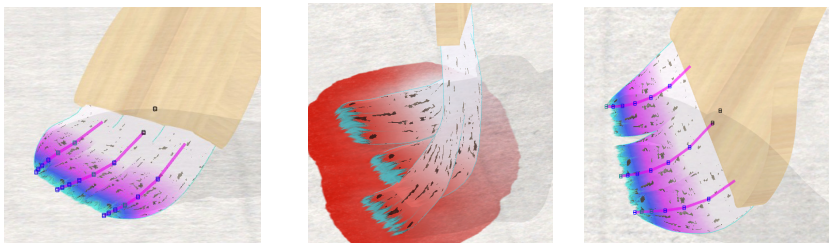


Figure 2.11: Left: Flat brush modeled with multi-spine skeleton. Middle: Unnatural splitting of a simulated flat brush. Right: Unnatural splitting of a simulated flat brush.

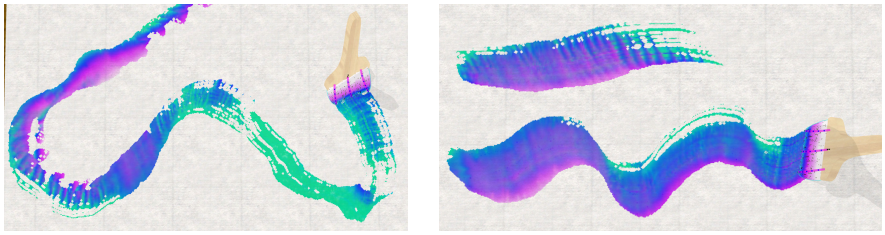


Figure 2.12: Sample strokes made with a simulated flat brush.

2.9 Results

Without paint flow simulation, the strokes made with our brush models are like those made with acrylic or non-impasto[3] style oil. Figures 2.13 – 2.16 show sample painting made with my brush model. Figure 2.17 shows sample strokes. Figure 2.18 demonstrates stroke playback function, with the original stroke in black and played-back strokes in red and green. I intentionally have the initially state of the brush different so that the played-back strokes are a bit different.

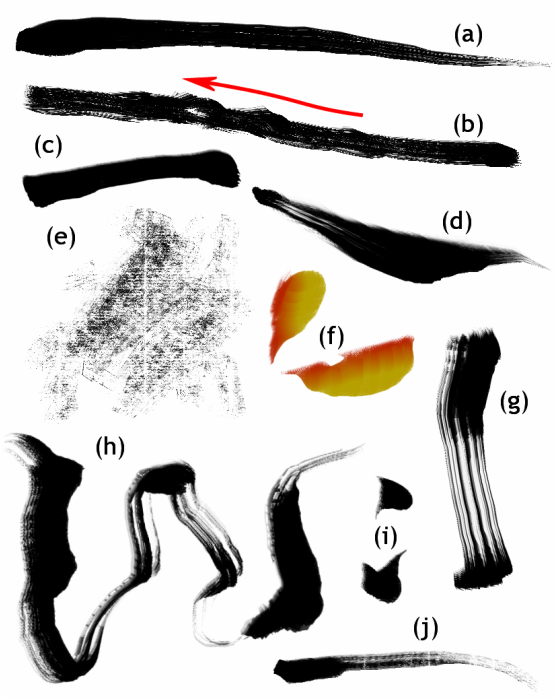
Figure 2.13: Sample painting painted with my brush model (res.: 1024 x 1024).

[3] Impasto refers to a style in oil painting in which thick layers of paint are generously laid on the canvas giving a 3D effect.

Figure 2.14: Sample painting by Paulo Purim (res.: 512 x 512).

Figure 2.15: Rocks painted with my brush model (res.: 512 x 512).

Figure 2.16: Flower painted with my brush model (res.: 512 x 512).



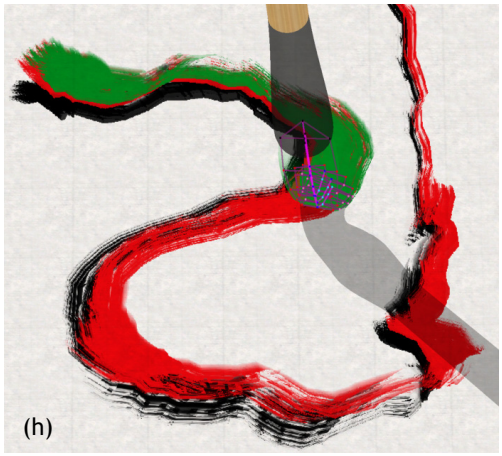


Figure 2.17: Sample strokes made with a round brush. (a) Central-tip. (b) Pushed; arrow indicates direction. (c) Biased-tip. (d) Blade-like calligraphic. (e) Rubbing with a dry brush. (f) Petals made with gradient loaded. (g) With a flattened brush. (h) With multiple tufts. (i) Calligraphic dots. (j) With a drying brush. (h) Playing back recorded strokes created with multiple tufts.

CHAPTER 3

WATER BASED PAINT FLOW SIMULATION

In this Chapter, I present my physically-based paint flow simulation technique. Background on the art media of Eastern ink and Western watercolor will first be given. I then review previous work on ink and watercolor simulations. After that, my simulation technique will be presented. I first describe how Eastern ink can be simulated with the Lattice Boltzmann method, and then go on to describe how such technique can be extended to simulate Western watercolor and an art form called 'suminagashi'. The ink simulation technique was first presented in [Chu & Tai 2005].

Our goal is to simulate water-based paint flow so that artists can paint in a spontaneous style on a computer interactively. Accomplishing this goal is challenging because of the complexity of the art medium and the real-time requirement. Apart from replicating existing artistic effects, we also hope to further develop the art by creating new digital effects, in particular those that carry the spirit of water – its fluidity.

3.1 Water-Based Paint Media

I love water-based paint. Water flows freely, and art media involving water often gives a sense of fluidity. I like the fact that painting in such media often requires balancing between *human control* and *natural outcome* to make a masterpiece.

3.1.1 Eastern Ink

In the spontaneous style of Eastern ink painting (or ink painting for short), artists utilize flexible brushes to create expressive lines and shapes, and exploit the interplay of ink and water to produce shades and patterns. To provide readers with the necessary background, we next discuss the art materials, the artistic effects related to ink dispersion, and the physical processes involved.

Eastern ink in its solid form is a mixture of soot and glue [Swider *et al.* 2003]. The solid is grinded together with water to get black liquid ink. The soot is composed of 10-150 nm carbon particles and dissolves readily in water. Because of their small sizes, the carbon particles seep into paper fibers easily, giving the most prominent dispersion effects in comparison with other (color) pigments. The glue functions as a dispersing agent to provide a stable liquid suspension of pigment particles and as a binder to fix the pigment during its application on paper. More glue in the ink makes the ink more viscous.

Ink dispersion is closely determined by paper absorbency; it is mainly the imbibition of water that causes the ink to flow through the paper fibers. Very thin and highly absorbent *Xuan* paper is commonly used to vividly pick up the special charm of ink dispersion. To reduce absorbency, the paper can be treated with *alum*.

3.1.2 Ink Painting Effects

For a faithful simulation, we need to model the essential effects of the art medium, which include:

- Feathery pattern (Figure 3.1(a)): When diluted ink is absorbed into the paper, delicate ink streaks that follow the direction of the water flow appear. This feathery quality is caused by the pigments being hindered or used up while the ink spreads;

- Light fringes (Figure 3.1(b)): When a stroke is made with diluted ink, water percolates on paper forming a light fringe around the stroke. If another stroke is painted over the first, the wet region of the paper will be less receptive to the new stroke, making the light fringe stands out;

- Branching pattern (Figure 3.1(c)): Branching patterns appear because the water flow is obstructed by irregularities in the paper or trapped ink ingredients. Artists may apply glue or other materials unevenly onto the paper to enhance this effect;

- Boundary roughening (Figure 3.1(d)): As ink percolates through paper, the wet-dry boundary is pinned at different points by the irregularities in the paper.

This is more prominent with concentrated ink, forming toes around the boundary;

- Boundary darkening (Figure 3.1(e)): At the pinned boundary of a wet ink mark, water is lost to the surrounding dry fibers via capillary attraction. This water evaporates immediately without expanding the mark, and the attraction induces a migration of pigments that darkens the boundary.

There are painting techniques that artists can use to produce a few of these effects at once, giving a dynamic feel of the art medium. One general technique is *ink-breaking*, in which the artist first paints a stroke with a certain ink concentration and, while it is still wet, paints another stroke over it with a different ink concentration. The two strokes then interact to break the monotony in each of them.

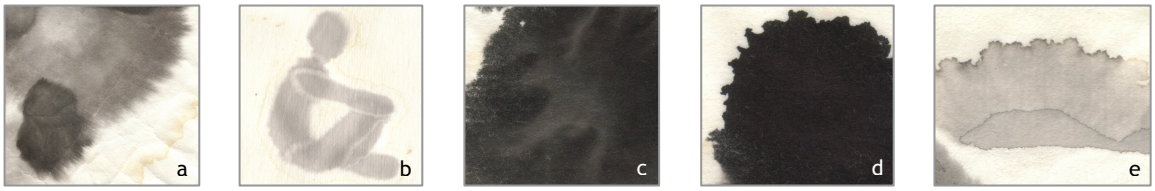


Figure 3.1: Real ink effects. (a) Feathery pattern. (b) Light fringes. (c) Branching pattern. (d) Boundary roughening. (e) Boundary darkening.

3.1.3 Physics of Ink Dispersion

In the graphics community, the term 'ink diffusion' has been traditionally used to describe ink dispersion in absorbent paper (e.g. [Guo & Kunii 1991; Lee 2001]). However, in physics, 'diffusion' refers to the random walk of particles towards the less concentrated regions. It is mathematically described by Fick's law. In effect, diffusion simply averages out any spatial difference in the concentration, making the concentration approach a constant with time.

In reality, the actual process of ink dispersion is a complex interplay between paper, water and ink constituents – a process that Fickian diffusion alone cannot describe adequately. Ink dispersion can be viewed as a two-part process: the percolation of water and the movement of pigments within the water. Water flows through the paper fibers in

low speed due to water pressure and capillary attractions. Voids in the paper, alum, and the accumulation of ink constituents all may impede the flow, inducing momentum exchange. When faced with obstacles, water branches into streams (Figure 3.2). The flowing water carries pigments with it. The dispersion of pigments in the water is mainly caused by the spatial difference in water velocity and the hindrance by paper fibers; pigment diffusion only plays a minimal role.

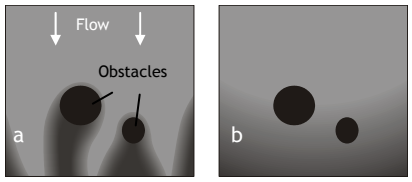


Figure 3.2: Impeded flow. (a) Modeled with momentum. (b) Just as diffusion or without momentum.

To better appreciate the above phenomena, I video record the ink dispersion process and play it back in fast-forward mode. I refer the readers to our accompanying video for sample footage (available at http://visgraph.cse.ust.hk/MoXi/). Realizing the complex nature of ink flow has motivated us to use a full-fledged fluid simulation to produce realistic ink effects.

Although very often we see traditional Chinese painting done in only black ink, colors are not missing in the Chinese painter's material set. Traditionally, Chinese colors are of two types: vegetation and mineral. Vegetation color pigments are extracted from plants and can be diluted with water. Colored minerals like Magnesium are used directly as the pigments.

3.1.4 Western Watercolor

There are basically two types of Western watercolor: transparent watercolor and gouache. We define gouache as color pigments mixed with opaque white pigments in a watercolor vehicle. Essentially, we can treat gouache as *opaque watercolor paint*. I refer the readers to [Curtis *et al.* 1997] or the website Handprint [Handprint Watercolor 2007] for a very nice examination of various artistic effects in watercolor.

3.1.5 Media Comparison

I was often asked about the difference between Eastern ink and Western watercolor. The main cause of differences is the paper used. In ink painting, very thin and absorbent paper is used, while in watercolor, thicker and less absorbent paper is used. Many artists treat the thinnest *Xuan* paper as the best paper because color laid on it are the most vivid. It is also the hardest to control moisture on such paper.

On very thin and absorbent paper, a small drop of ink would be imbibed to a large area. Such imbibition carries pigments through a long way and streaks like those shown in Figure 3.3 are only possible by such imbibition. Note that ink flow mostly through the paper fibers and thus tilting the paper has little effect on the flow. On the other hand, paint flow in watercolor mainly occurs on the paper surface so tilting the paper will cause bulk paint movement. Pigments also tend to migrate to wet boundary or clump on the paper surface (Figure 3.4). The difference between Eastern ink and Western watercolor is summarized in Table 3.1.

	Eastern Ink	**Watercolor**
Water Flow	Mostly thru fibers	Mostly on surface
	Impeded by glue, alum	Perturbed by paper bump
Pigments	Small	Large

Table 3.1: Differences between Ink and watercolor.

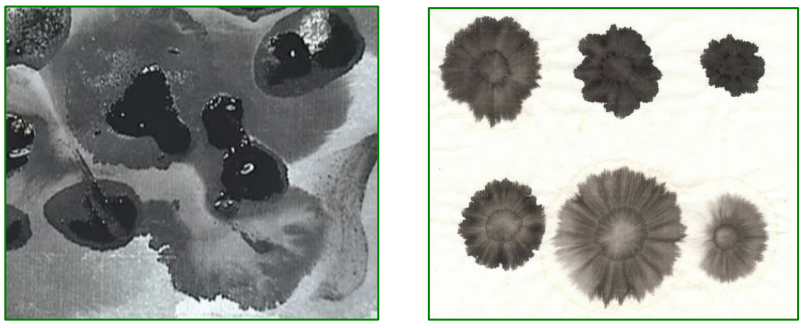


Figure 3.3: Flow patterns unique to ink painting. Left: Flow patterns produced by flushing ink with water through deposited ink. Right: Patterns obtained by first depositing concentrated ink drops, and then water drops at the same spots in different time intervals.

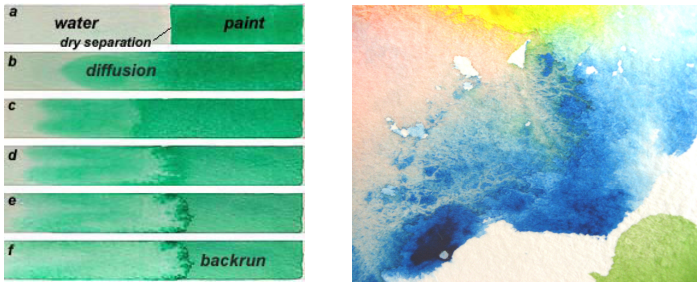


Figure 3.4: Characteristic patterns in watercolor. Left: Diffusion and Backrun in watercolor. Right: Granulation. (Left image from www.handprint.com © 2007 Bruce MacEvoy)

3.2 Related Work

Here I briefly review some of the most relevant related work on the simulation of Eastern ink and Western watercolor. Because the simulation of water-based paint flow involves fluid dynamics, I also review related simulation techniques in the computer graphics context.

3.2.1 Eastern Ink

Guo and Kunii [1991] proposed the first model specifically for Eastern ink dispersion in paper. Assuming that ink consists of pigment particles of different sizes, they modeled ink spreading as a one-dimensional filtering process. Later, Kunii *et al.* [2001] used partial differential equations (PDE) in an attempt to better describe the phenomenon. Their PDE's for water spreading in paper and pigment movement within water are essentially Fick's law of diffusion. However, these PDE's could only produce blurry images without the interesting flow patterns observed in reality.

Other researchers [Zhang *et al.* 1999; Yu *et al.* 2002] used cellular automation to simulate ink dispersion. They essentially treat ink flow as a relaxation process without momentum exchange. Without considering momentum, these methods are only suitable for situations where ink mixes to form blurry images. Yet other research efforts [Lee 2001; Guo & Kunii 2003] processed only the fronts of the spreading strokes to speedup rendering, but this also limits the range of producible effects.

3.2.2 Western Watercolor

Curtis *et al.* [1997] did some very successful simulations of Western watercolor paint. Their model simulates the flow of water in two parts: on the paper surface, and through the paper fibers. They solve the Navier-Stokes (N-S) equations for the on-surface flow and use a cellular automation for the capillary flow through fibers. The latter is used only to produce an effect called back-run.

Curtis *et al.*'s model does not simulate Eastern ink effects well. First, they use a fixed wet-dry boundary for their N-S equation solver, making shape evolution of ink marks impossible. Using their capillary flow for boundary expansion would produce unrealistic spreading because the two flow processes are only loosely coupled. In particular, the water supply from the surface to the capillary layer is not tracked, so a larger puddle of ink on the surface would not result in a larger blob. Second, their model omits medium permeability, which is essential for generating subtle ink patterns.

Laerhoven *et al.* [2004] performed watercolor simulations similar to Curtis *et al.* [1997] on a grid of processing units. They employed a semi-Lagrangian method [Stam 1999] for faster simulation. A frame rate of 25 frames per second (fps) for a canvas size of 256^2 with 6 processors is reported. Laerhoven and Reeth [2005] later also did a GPU implementation of the fluid solver. A frame rate of 20 fps for a canvas size of 800×600 with a tiling size of 32^2 running on a Geforce 6800 graphics card is reported.

3.2.3 Fluid Simulation

In the past decade, graphics researchers have adopted numerical fluid simulation to add realism to computer generated animations (e.g. [Harris 2003]). They typically followed the traditional approach of beginning with a macroscopic description of the fluid, namely, the Navier-Stokes (N-S) equations. Because the N-S equations are hard to solve, various approximate numerical techniques have been proposed [Ferziger & Peric 1999]. One popular method among the graphics researchers is the scheme presented by Stam [1999]. Real-time applications favor this scheme for its ability to take large simulation time step stably. Yet, too large steps result in visible artifacts [Song *et al.* 2005], hence real-time simulations often have to be done at 256^2 or lower resolutions.

Stam's scheme has also been implemented on the GPU for acceleration [Harris 2003]. However, this scheme has to solve two Poisson equations: one for the pressure (often the performance bottleneck) and one for the viscous diffusion. Solving the Poisson equations involves global operations, making the method not intrinsically suitable for parallel GPU processing.

An increasingly popular alternative approach for fluid simulation is the lattice Boltzmann method (LBM) [Succi 2001; Wei *et al.* 2004]. Instead of starting with a macroscopic description of the fluid, the LBM models the physics of fluid particles at a mesoscopic level. We base our ink flow simulation on the LBM for its advantages detailed in the next section.

3.3 Why Lattice Boltzmann?

I surveyed various fluid simulation techniques and chose to simulate water percolation using the LBM. Advantages of the LBM over the traditional N-S equation solvers include [Yu *et al.* 2003]: (1) it does not involve Poisson equations, (2) all operations are simple and local, and (3) easy to incorporate physics that is hard to describe macroscopically. Items (1) and (2) facilitate very efficient implementation on parallel graphics hardware, while item (3) eases the incorporation of extra physics for new paint effects. Fluid compressibility, the main feature of the LBM that gives it a performance advantage, however, also creates an issue for simulating incompressible flow: the flow speed must stay low to keep the compressibility effect negligible. Fortunately, this is not a problem for us, since ink flows slowly through fibers.

In my opinion, LBM as a general fluid simulation method is often overlooked and it deserves more appreciation. The LBM algorithm is much simpler than 'traditional' methods in solving the N-S equations. In fact, LBM makes me really appreciate the beauty of mathematics. In a review for the book [Succi 2001], Boghosian [2003] said:

> *'The typical reaction of long-time computational fluid dynamics practitioners when encountering lattice Boltzmann algorithms for the first time is often something like, "that can't possibly work. It's too easy."'*

This kind of reaction is probably common in our graphics community too. Mark Harris in a SIGGRAPH 2004 course [Harris 2004a] mentioned LBM under the heading of 'Approximate Methods', and next described more complicated Navier-Stokes solver [Stam 1999] as 'less *ad hoc* method'. I did not attend the course, but looking at the course notes, I expect lots of people would get the misconception of LBM being 'not ready for primetime'. I would like to say that the simplicity in LBM does not imply it is *ad hoc* or less accurate than other N-S solvers. In fact, the LBM is rigorously based on kinetic theory [Succi 2001]. Both LBM and Stam's scheme are approximate by definition, as both are at least discretized at some point. However, from my understanding of Stam's method and tinkering with Harris' GPU implementation, I would say that in real-time fluid simulations based on Stam's scheme, much accuracy is actually given up in favor of real-time response.

But how can LBM perform so much faster if it is not inaccurate? Well, LBM actually resembles the Artificial Compressibility (AC) method developed by Chorin [1967] over thirty years ago [He *et al.* 2002]. The AC method makes large time step possible by allowing weak compressibility in the simulation fluid. Intuitively, we can understand this using an analogy of a box filled up with marble balls. If you push one ball, all other balls are also pushed around to make room for the new position and to fill up the old position of the ball pushed by you. This global rearrangement, which translates to solving a Poisson equation, is complicated to resolve. If now the balls are compressible, the transmission of shock need not be strictly immediate and thus is much easier to resolve. This artificial compressibility treatment is 'pragmatic rather than physical' [He *et al.* 2002]. In practice, with comparable accuracy, LBM requires the same or less computation than 'traditional' finite-element or finite-volume based N-S solvers for non-turbulence flow [Geller *et al.* 2006]. And, speaking of performance, don't forget that LBM has yet one more advantage of being intrinsically suitable to run on parallel GPU.

3.4 Lattice Boltzmann Equations

The Lattice Boltzmann formulation is fully discrete in time and phase space, which is a simplified version of the continuous Boltzmann equation. For a gentle introduction, I

recommend the article by Chen *et al.* [1994]. I adapt the basic LBM to incorporate various features needed for the special case of percolation (described in Section 5.3). The main idea of the LBM is to model fluid dynamics using a simplified particle kinetic model. This approach divides the simulation domain into a regular lattice. At each lattice site **x** and time *t*, the fluid particles moving at arbitrary velocities are modeled by a small set of particle distribution functions $f_i(\mathbf{x}, t)$, each of which is the expected number of particles moving along a lattice vector $\mathbf{e}_i$. We use a standard square lattice model for 2D flow, called D2Q9 (Figure 3.5). In this model, there are nine lattice vectors: $\mathbf{e_1}$, $\mathbf{e_2}$, $\mathbf{e_3}$, $\mathbf{e_4}$ pointing to the four nearest neighbor sites, $\mathbf{e_5}$, $\mathbf{e_6}$, $\mathbf{e_7}$, $\mathbf{e_8}$ pointing along the diagonals to the next nearest sites, and the zero vector $\mathbf{e}_0$ corresponding to the stationary particles. During each time step Δt, two operations are performed at each lattice site: (1) streaming f_i's to the next lattice site along their directions of motion, and (2) colliding f_i's that arrive at the same site. The collision redistributes f_i's toward their equilibrium distribution functions $f_i^{(eq)}$. The two operations of streaming and collision are mathematically described by the Lattice Boltzmann Equations (LBE):

$$f_i(\mathbf{x}+\mathbf{e}_\mathbf{i}\Delta t, t+\Delta t) = (1-\omega) f_i(\mathbf{x}, t) + \omega f_i^{(eq)}(\mathbf{x}, t) \quad , \tag{1}$$

where ω is the relaxation parameter. There are numerous variants of the LBM flow model. We employ the 'incompressible' variant of He and Luo [1997], which can minimize the compressible effect inherent in the LBM. In this model, the equilibrium distributions $f_i^{(eq)}$ are

$$f_i^{(eq)} = w_i \left\{ \rho + \rho_0 \left[\frac{3}{c^2}\mathbf{e}_\mathbf{i}\cdot\mathbf{u} + \frac{9}{2c^4}(\mathbf{e}_\mathbf{i}\cdot\mathbf{u})^2 - \frac{3}{2c^2}\mathbf{u}\cdot\mathbf{u} \right] \right\} , \tag{2}$$

where c equals Δx/Δt, Δx is the lattice spacing, w_i are constants determined by the lattice geometry, ρ and **u** are fluid density and velocity, respectively, and ρ_0 is a predefined average fluid density. For simplicity, we set Δx = Δt = c = ρ_0 = 1 in our simulation. The constants w_i are set as 4/9 for i = 0, 1/9 for i = 1,2,3,4, and 1/36 for i = 5,6,7,8. Fluid density and velocity at each site are given by

$$\rho = \sum_{i=0}^{8} f_i \quad , \qquad \mathbf{u} = \frac{1}{\rho_0} \sum_{i=1}^{8} \mathbf{e}_i f_i \ . \qquad (3), (4)$$

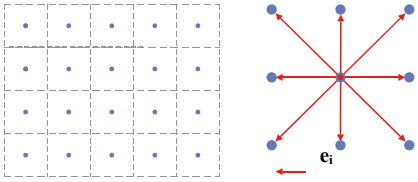


Figure 3.5: The D2Q9 lattice model. Left: The lattice. Right: The nine lattice vectors of a lattice site.

3.5 Simulation Overview

Physicists have proposed various models for the sub-problems of imbibition [Davis and Hocking 2000], porous media flow [Adler 1992], hydrodynamic dispersion [Sun 1996], and boundary roughening [Alava *et al.* 2004]. Only some of these models are practical for our application owing to modeling efficiency, leaving various holes to be filled in a good solution. My challenge is therefore to design a unified model that can capture the essence of the physical processes involved so as to produce the desired results, and yet is computationally efficient to work in real time. An additional requirement is that our ink simulation has to work well together with an accurate brush dynamics simulation for intuitive deposition of ink from brush to paper.

Like previous graphics research (e.g. [Curtis *et al.* 1997]), we do not aim to do strict scientific simulations. I bring in physics only to improve the digital painting experience. Our ink simulation is based on a three-layer paper model. The layers are *surface*, *flow* and *fixture*. Ink is first deposited onto the surface layer. For simplicity, we assume that the ink flows only in the flow layer and not in the surface layer. The ink on the surface gradually seeps into the flow layer, where water percolates through paper and ink constituents are advected by the water. Finally, as the ink dries, ink constituents in the flow layer are moved slowly into the fixture layer.

I develop an LBM-based fluid flow model to simulate the water percolation. I also couple this flow model with a simple method for simulating the movement of ink constituents. For realistic ink deposition from brush to paper, we employ our existing physically-based brush dynamics model presented in Chapter 2. Among the three simulation parts – *ink deposition*, *water percolation*, and *movement of ink constituents* – the simulation of water percolation is the most important for capturing the dynamic nature of the art medium.

The quantity fields used in our ink flow simulation (e.g. water velocity) are discretized spatially on the lattice in the LBM. These data fields are updated iteratively using the GPU. The CPU is dedicated to the brush simulation. In designing our ink simulation algorithms, we assume a stream-processing model in modern GPU architecture [Mark *et al.* 2003]. All our discretized data fields are stored as texture maps. In the description of our methods below, we omit the spatial indices to data fields (like ρ) when we refer to the quantity at the current lattice site as in a data streaming model.

3.6 Ink Deposition

The amount of ink deposited onto the surface layer is determined by the brush footprint and the saturation of water in the brush and in the paper. By simulating the variation in paper receptivity caused by saturation, we can produce the effect of a light stroke fringe (Figure 3.6(b)). Ink is supplied from the surface to the flow layer according to the capacity of the paper fibers. The surface layer stores excess ink not yet absorbed, which acts as a reservoir.

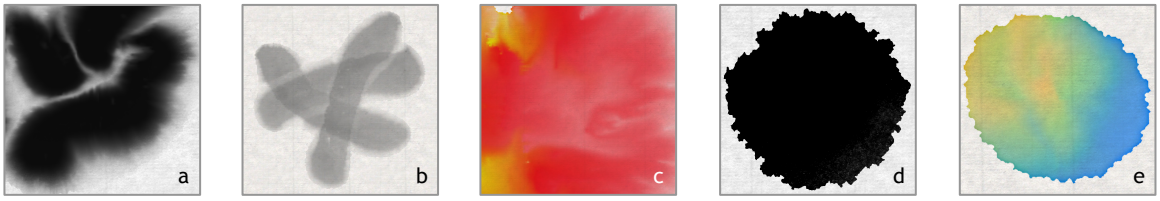


Figure 3.6: Digital ink effects created using our system similar to the real effects illustrated in Figure 3.1.

To generate the brush footprints, we use the brush model described in the last Chapter. To model variable paper receptivity, each pixel in the footprint is masked by the value $max(1-\rho/\lambda, m)$, where ρ is the water density (amount of water) at the corresponding lattice site in the flow layer, λ is a receptivity parameter, and m is a base mask value. The amount of water supplied from the surface to the flow layer φ is taken as $clamp(s, 0, \pi-\rho)$, where s denotes the amount of water on the surface, π is the capacity of the paper fibers, and $clamp(x, min, max)$ is a function that returns the clamped value of x against the limits of min and max. In our simulation, we use $\pi = 1$, $0.3 \le \lambda \le 1$, and $m = 0.1$. After φ is determined, s and ρ are updated accordingly. Ink constituents carried by the water that seeps to the flow layer are also moved as described in Section 5.4.1.

3.7 Water Percolation

I apply the LBM to model the water movement in the flow layer. The original Lattice Boltzmann equations describe fluid flows without considering medium permeability and free boundary evolution. To deal with the more complex situation of percolation, I made several modifications to the basic Lattice Boltzmann equations: variable permeability, advection modulation, boundary evolution, and uneven evaporation.

3.7.1 Permeability and Viscosity

Variable permeability is one key element in our model that makes the creation of interesting flow patterns possible. Recall that the water flow is impeded by various irregularities in the paper. This impediment can be modeled as the permeability of the simulation domain. Like Dardis and McCloskey [1998], I use fractional (rather than Boolean) values to represent the permeability of each lattice site. The permeability is realized by blocking the streaming process; thus we associate each site with a blocking factor κ. Varying κ allows us to model a wide range of media, from those that allow no water movement to those where perturbation in ρ causes ripples.

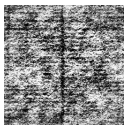
Specifically, I use the half-way-bounce-back scheme [Succi 2001] during the streaming process to simulate the variable permeability. The blocking is performed as if the link to each neighboring site is partially blocked with a blocking factor $\bar{\kappa}_i$. Rather

than having $\bar{\kappa}_i$ equal to the blocking factor of the destination site as in [Dardis and McCloskey 1998], we let $\bar{\kappa}_i$ be the average of the blocking factors of the two linked sites. This guarantees the same amount of blocking in both streaming directions, hence conserving the momentum and density. The streaming step with bounce-back is mathematically described by

$$f_i(\mathbf{x}, t+1) = \bar{\kappa}_i(\mathbf{x})\, f_k(\mathbf{x}, t) + (1-\bar{\kappa}_i(\mathbf{x}))\, f_i(\mathbf{x}-\mathbf{e}_\mathbf{i}, t) \quad , \tag{5}$$

where f_k is the distribution function pointing in the opposite direction of f_i.

Two basic factors that determine the paper permeability are the voids in the fibers and the alum deposited. I store such information in texture maps. Because most types of Eastern painting paper are very thin and semi-transparent, we can obtain the paper thickness patterns simply by scanning the paper against a dark background. We use the normalized thickness images as *grain textures* (a sample is shown on the right), which serve as patterns for the paper grain or voids in the fibers. *Alum textures* that record alum deposition can be generated procedurally (e.g. with some random dots). Precisely, the blocking factor κ at each site is defined as



$$\kappa = k_1 + k_2 \cdot G + k_3 \cdot A + k_4 \cdot g + k_5 \cdot h\,, \tag{6}$$

where G and A are values of the grain texture and the alum texture, respectively, k_i are weights that define the blocking, g is the glue concentration in the flow layer, and h is the ink constituent accumulation in the fixture layer.

Another factor that affects the ink flow is the viscosity. Artists sometimes add extra glue to the ink to limit its spread. For a good control, we want the flow to vary from being completely stationary when glue = 1 to normal when glue = 0. In the LBM model, viscosity is given by $(1/\omega - 1/2) / 3$, assuming $\Delta t = c = 1$. Lowering the relaxation parameter ω increases the viscosity. However, we found that modulating ω with glue does not give the desired behavior: the flow cannot remain still when glue = 1 since f_i's from the areas with less glue continue to stream in.

Our solution is to modulate κ with glue, which forms part of our formulation for permeability. The modulation of viscosity with the relaxation parameter ω is still used for adjusting the viscosity globally. In our simulation, $\omega = 0.5$ normally, and reaches 1.5 for simulations of more fluent flow.

3.7.2 Free Boundary and Advection

As water percolates, free boundaries exist between the air (in the fibers) and the water. In fluid dynamics literature, air and water are referred to as two *phases*. Since the effect of air is negligible, we use only a single-phase model, for water. Devising a single-phase free-boundary LB model is, nevertheless, not straightforward. The LBM model was originally designed for situations where the simulated fluid fills the whole domain, with small local deviation in velocity and density (within 10 to 20 percent) from the mean. If we use the fluid density to represent how much a site is filled (zero density for empty sites), applying the original LBE would cause negative density in certain sites. This is because the advection built into the LBE carries density away even from sites with near-zero density. Existing single-phase models deal with negative densities by simply averaging among neighboring sites [Thuerey 2003] or by extrapolation [Ginzburg and Steiner 2003].

To avoid the unphysical situation of negative density from happening altogether, we modify the basic LBE to reduce the strength of the advection when the density is low. Our rationale is that the advection of water drops in less saturated areas. Specifically, we add a weight ψ to the terms responsible for advection:

$$f_i^{(eq)} = w_i \left\{ \rho + \rho_0 \, \psi \left[\frac{3}{c^2} \mathbf{e_i} \cdot \mathbf{u} + \frac{9}{2c^4} (\mathbf{e_i} \cdot \mathbf{u})^2 - \frac{3}{2c^2} \mathbf{u} \cdot \mathbf{u} \right] \right\} \tag{7}$$

$$\psi = smoothstep(0 \ , \alpha, \ \rho) \ , \tag{8}$$

where α is a user parameter for adjusting this effect. It can be readily checked that the conservation of total water density still holds. The range $0.2 \le \alpha \le 0.5$ works well in our simulation.

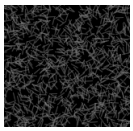
We define a *boundary site* to be a wet lattice site (i.e. $\rho > 0$) with at least one dry site among its eight neighbors. All single-phase free-boundary LBE models require interfacial boundary conditions to determine the particle distribution functions f_i of the boundary sites, including those that are streaming inward from outside the boundary. In our model, this is taken care of by our full-bounce-back pinning mechanism, which is detailed next.

3.7.3 Boundary Pinning and Roughening

Boundary roughening is caused by the spreading front being pinned at different points [Alava *et al.* 2004]. A front is depinned when there is enough water pressure to overcome the pinning. In reality, the process involves complex interaction between fiber web and ink constituents, and other factors (e.g. evaporation).

We use simple local rules to model pining and depinning, which can be efficiently integrated into the LBE. We say a lattice site is a *pinning site* if it is dry and the water density ρ at each of its eight neighbors is below a threshold. The four nearest neighbors share the same threshold, denoted by σ. We set the threshold for the four next nearest neighbors as $\sqrt{2}\,\sigma$ to compensate for geometric difference of the links. To effect the pinning, we overwrite the blocking factor κ at all the pinning sites by a large value to fully block all their neighbor links.

For the toe patterns found in real ink marks made with concentrated ink (Figure 3.6(d)), we introduce one more texture map called *pinning texture* to model the effect of paper disorder. The map is generated by sprinkling light line segments on a dark background (a sample is shown on the right). Modulating the threshold σ with this map gives the effect of easier ink flow at certain locations and directions. We have found this to work well in producing the desired patterns (Figure 3.6(d)) when we also model boundary evaporation, which is described in the next subsection. This evaporation avoids the toes from simply taking the shapes of the overlapping line segments in the pinning texture.



We also have σ dynamically depend on the glue concentration in the flow layer, denoted by g, and the ink constituent accumulation in fixture, denoted by h. Precisely, σ is defined as

$$\sigma = q_1 + q_2 \cdot h + q_3 \cdot lerp(G, P, smoothstep(0, \vartheta, g)) \ , \tag{9}$$

where G and P are values of the grain texture and the pinning texture, respectively, q_i are weights that define the roughening behavior, and ϑ is a parameter controlling the effect of the glue concentration on the appearance of toes. The function *lerp*(a, b, f) returns the linear interpolation (1- f) a + f b. *Smoothstep()* is a built-in function found in both shading languages nvidia Cg and OpenGL GLSL. Mathematically, *smoothstep*(*min, max, x*) is evaluated as -2*((*x-min*)/(*max-min*)) ^{3} + 3*((*x-min*)/(*max-min*)) ^{2}. For smooth spreading fronts (Figure 3.14(b)), we can simply set $q_2 = q_3 = 0$.

3.7.4 Evaporation

As a stroke dries, a small extra loss of water occurs at the pinned boundary via capillary attraction. This induces a migration of pigments towards the boundary resulting in a darkened edge.

We model this by having different evaporation rates for the pinned boundary sites and the rest of the wet sites. We do this by reducing the water density ρ at a rate of ε_s as we update ρ using Eq. (3), and reducing those f_i's that bounce back due to pinning at a rate of ε_b during the streaming step. This process is similar to the way Curtis *et al.* [1997] produced their edge-darkening effect, except that they use a smooth spatial falloff to vary the evaporation rate. Our simulation uses $0 \le \varepsilon_s \le 0.005$ and $\varepsilon_b = 5 \times 10^{-5}$.

3.8 Pigment Advection

Since pigments and glue are moved likewise, we will only describe how pigments are moved. Concentrations of different pigments are stored in different channels of RGBA textures. We will use a bold symbol (e.g. $\mathbf{p_f}$) to denote the array of concentrations at a site. In our simulation, the movement of pigments can be divided into three parts: *supply*, *advection*, and *fixture*.

3.8.1 Pigment Supply

The ink deposited on the surface layer serves as a reservoir that provides ink supply to the flow layer. After determining the amount of water supplied from the surface, φ (Section 3.6), the pigments in the flow layer is updated according to the ratio of ρ to φ as: $\mathbf{p_f} \leftarrow (\mathbf{p_f}\,\rho + \mathbf{p_s}\varphi) / (\rho + \varphi)$, where $\mathbf{p_f}$ and $\mathbf{p_s}$ denote the pigment concentrations in the flow layer and the surface layer, respectively.

3.8.2 Pigment Advection

The movement of pigments in the flow layer is governed by hydrodynamic dispersion, in which advection dominates diffusion. In our advection scheme, we first differentiate sites that are already wet from those that are becoming wet in the current time step. For the latter, we derive $\mathbf{p_f}^*(\mathbf{x})$, the pigment concentrations newly advected to site $\mathbf{x}$, as:

$$\mathbf{p}_\mathbf{f}^*(\mathbf{x}) = \frac{1}{\rho}\sum_{i=1}^{8} f_i\, \mathbf{p_f}(\mathbf{x}-\mathbf{e_i}) \tag{10}$$

For the former, $\mathbf{p_f}^*(\mathbf{x})$ is obtained by tracing the velocity backward as in the *method of characteristics* [Sun 1996]: $\mathbf{p_f}^*(\mathbf{x}) = \mathbf{p_f}(\mathbf{y})$, where $\mathbf{y} = \mathbf{x} - \mathbf{u}(\mathbf{x})$. Since our data fields are stored as textures, it is very efficient to use hardware interpolation to sample $\mathbf{p_f}(\mathbf{y})$. But this interpolation would give a wrong $\mathbf{p_f}^*(\mathbf{x})$ when $\mathbf{y}$ is within half a lattice spacing from the boundary. The reason is that $\mathbf{p_f}(\mathbf{y})$ would be the result of a cross-boundary interpolation, but any $\mathbf{p_f}$ outside the boundary should never contribute to $\mathbf{p_f}^*(\mathbf{x})$. Our solution is simply to use $\mathbf{p_f}^*(\mathbf{x}) = \mathbf{p_f}(\mathbf{x})$ when this happens.

Finally, we update $\mathbf{p_f}(\mathbf{x})$ with an interpolation between $\mathbf{p_f}^*(\mathbf{x})$ and the old $\mathbf{p_f}(\mathbf{x})$ to model the hindrance of pigments by the irregularities in paper according to the pseudo-code:

```
Proc SimulateHindrance(p_f, p_f*, γ, ς, u) {
    γ* ← lerp(1, γ, smoothstep(0, ς, |u|))
    p_f ← lerp(p_f*, p_f, γ*)
}
```

Here γ is the hindrance rate, ς is a parameter for blocking the advection when the flow speed is low.

3.8.3 Pigment Fixture

Pigments in the flow layer are gradually transferred to the fixture layer as the ink dries. Given the fact that real dried ink mark cannot be washed away by water, we assume that the transfer is a one-way process. For realistic pigment behaviors, we want to satisfy the following conditions: (1) the transfer rate is higher when the strokes become drier, (2) the transfer rate is higher when the glue is more concentrated, and (3) all pigments are settled when a stroke dries.

We devise a simple pigment fixture algorithm that satisfies the above conditions. It updates the pigment concentrations in the flow layer and the fixture layer, denoted by $\mathbf{p_f}$ and $\mathbf{p_x}$, respectively, according to several parameters: ρ and ρ', the water density in the flow layer in the current frame and the last frame, respectively; g, the glue concentration in the flow layer; η, a base fixture rate; μ and ξ, parameters for modulating the fixture rate by dryness and glue, respectively. The algorithm expressed in pseudo-code reads:

```
Proc SimulateFixture(p_f, p_x, ρ, ρ′, g, η, μ, ξ) {
wLoss ← max(ρ′ − ρ, 0)
if wLoss > 0 then
    FixFactor ← wLoss / ρ′
else
FixFactor ← 0
μ* ← clamp(μ + ξ × g, 0, 1)
FixFactor ← max(FixFactor × (1 − smoothstep(0, μ*, ρ)), η)
p_x ← p_x + FixFactor × p_f
  p_f ← p_f − FixFactor × p_f
}
```

3.9 GPU Implementation

We have implemented a paint system, named *MoXi*, based on our ink dispersion model using OpenGL and the Cg shading language [Mark *et al.* 2003]. The simulation operations are implemented in fragment programs running on the GPU. The simulated quantities are stored as textures, which are updated with the result of the fragment programs. For manipulating the virtual brush, our system supports graphics tablet (Figure 1.8), which is popular among digital artists.

GPU Processing: Currently, we limit our dispersion simulation to three pigments (colors) at a time so that we can fit pigments and glue concentrations into one RGBA texture. A large gamut of colors, however, is still available by mixing the three pigments (e.g. cyan, magenta, yellow). During each time step, we perform six texture updates for the LBM flow simulation, and another six texture updates for moving the pigments and glue. The boundary trimming requires one more texture update for I'. Thanks to the simplicity of the method, the six fragment programs for the LBM operations have an average assembly instruction count of only 29.8. Further details are presented in the supplementary material.

Performance: We conducted performance test on a machine with an Althlon XP 2600+ CPU and a GeForce 6800Ultra GPU with 256MB video memory. We denote the ratio of the dimensions of I to those of I' by *d-scale*. For real-time generation of output at 1536^2 with a simulation resolution of 512^2 and a *d-scale* of 3, we are able to have an overall system frame rate (including the high-quality rendering techniques described in Section 6) of 44 frames per second. More performance data are shown in Table 3.2.

Sim. Resol.	*d-scale*	I' Update Interval (fr.)	fr./sec.
256^2	1	-	70
512^2	1	-	48
512^2	3	300	44
512^2	3	1	32
512^2	4	300	42

Table 3.2: Performance of our system. Frame rates are measured when there is no concurrent brush deformation.

3.9.1 Data Packing

All data fields (like water density, velocity) needed for the ink simulation are stored as RGBA textures. For the LBM flow simulation, we use 4 textures listed in Table 3.3. We call these *simulation textures.*

Texture	Contents
1. VelDen	[u, v, wf, seep]
2. Misc	[blk, f0, lwf, ws]
3. Dist1	f[N, E, W, S]
4. Dist2	f[NE, SE, NW, SW]

Symbol Descriptions	
u, v	Water velocity
wf	Water density in flow layer
lwf	Water density in flow layer in the last iteration
ws	Water amount on surface layer
seep	Amount of water seeping from surface layer to flow layer
blk	Blocking factor
f0	Distribution function for stationary particles
f[N, E, W, S]	Dist. functions towards nearest neighbors
f[NE, SE, NW, SW]	Dist. functions towards next nearest neighbors

Table 3.3: Texture data packing.

3.9.2 Texture Update

Each of the above simulation textures is updated by rendering the paper geometry to a pixel buffer using a fragment program that performs the needed LBM operations. The content of the pixel buffer is then copied to the destination texture. The six texture updates for the LBM simulation are listed in Table 3.4.

Texture Update	Output Texture	Operations
1.	Misc	Derive f0, blk Deposit or update ws Save wf to lwf
2.	Dist1	Collide f[N, E, W, S]
3.	Dist2	Collide f[NE, SE, NW, SW]
4.	Dist1	Stream f[N, E, W, S]
5.	Dist2	Stream f[NE, SE, NW, SW]
6.	VelDen	Derive u, v, wf, seep Receive water from surface

Table 3.4: Texture updates for the LBM simulation.

For ink deposition, we apply fragment programs that perform the necessary operations on the brush tuft geometry. During the application, a small part of the brush tuft penetrates the virtual paper, giving the brush footprint (Section 2.7). The brush tuft geometry is stretched as shown in Figure 3.7 [Wloka and Zeleznik 1996] to give a continuous stroke (rather than instances of brush geometry at discrete times given by the brush dynamics simulator). Both the penetrating part of the stretched tuft and the paper geometry are rendered to the pixel buffer simultaneously so that deposition does not need extra rendering passes. However, we need to have the shader code for doing ink flow simulation on paper also called from the fragment programs for the brush geometry so that we can have the ink flow on paper still working for the portion of the paper covered by the brush.

It should also be noted that an alternative method for generating the 'stretched' footprint is used by Baxter [2004]. He generates a deposition map by stamping the brush footprint at one instance multiple times alone the distance the brush traveled between two consecutive time frames. In comparison, our method has the advantage of saving a separate render pass and texture memory for generating a separate deposition map but more shader code execution is needed on the brush geometry in our method as described in the last paragraph. However, when I later implement video input for the brush .footprint, I realized that a deposition map is needed anyway to consume the video input frame.

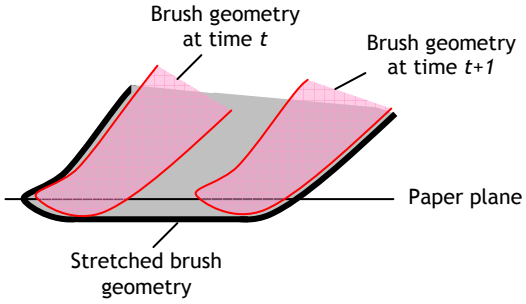


Figure 3.7 Stretching brush geometry for deposition.

3.9.3 Hardware Precision

Our ink flow simulation algorithm requires interpolation of texture values and in terms of computational efficiency, this is best obtained via hardware texture interpolation. The latest generation of GPU (e.g. nvidia Geforce 8 series) supports full 32-bit floating point data with bilinear texturing filtering. The computation in the fragment programs is done with 32-bit float precision, and I provide the option to switch between using 16-bit or 32-bit floating point texture. Using 16-bit float texture instead of 32-bit lowers the memory consumption and allows my program to run on the previous generation of graphics card. My current Windows-platform implementation still uses copy-to-texture model of texture update rather than render-to-texture [Wynn 2002]. Using 16-bit also makes the copying of data faster.

I encountered a few precision issues when using the 16-bit and/or 32-bit float textures. The first issue lies in the transfer of pigments from the flow layer to the fixture layer (Section 3.8.3). I found that pigments are lost during the transfer, with the destination texture not getting the correct amount. One likely cause is the hardware round-off/truncation procedure for the 16-bit float texture writes. I solve the issue by manually quantizing the transferred pigment amount with a quantization step of 1/2048 (determined empirically). This happens with both 16-bit and 32-bit float textures.

Another issue is in the hardware interpolation of float textures as needed in the generation of high quality output (Section 4.1.1). Some artifacts appear in the generated higher resolution output I': the pixels in I' that are supposed to be interpolated from the pixels in I were lighter than what they should be. I suspect this is caused by the hardware interpolation giving values a bit smaller than they should be. These small errors would not be noticeable if the interpolation is done only once, but become visible when accumulated. We solve this problem by reducing the number of updates to I'. In our implementation, we only update I' every 200-400 time steps. This happens with both 16-bit and 32-bit float textures.

One last issue is that when ink flows for some time, the ink simply gets paler (Figure 28). I obtain less paling with 32-bit float textures. It is also interesting to see that the same flow simulation would produce slightly different outcome as shown in Figure 3.8.

Since GPU hardware is still evolving and it is not easy for us to know exactly what happen in the hardware (GPU makers may use approximation to speedup rendering), I will leave these problems alone at the moment.

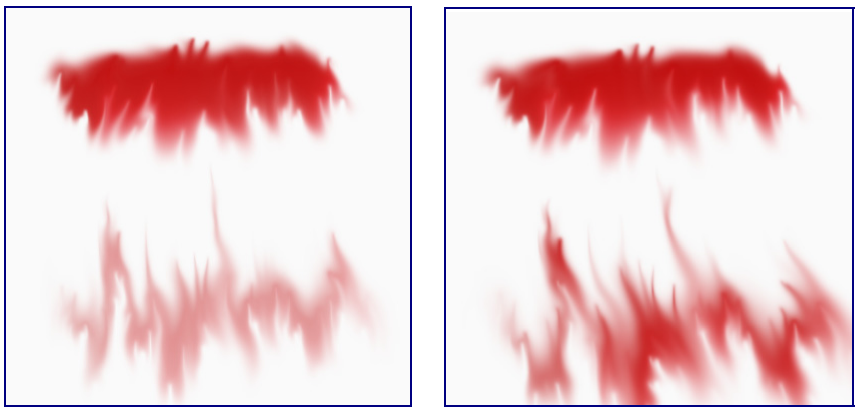


Figure 3.8. Comparison between different floating-point data precisions.
Left: Ink marks simulated with 16-bit. Right: Less paling with 32-bit.

3.9.4 Use of Infinity

We use the fact that any interpolation between a finite value and infinity is infinity to facilitate efficient processing. In our pigment advection scheme, we have to check if the pigment concentrations at the source point $\mathbf{p}_f(\mathbf{y})$ are cross-boundary interpolations (Section 3.8.2). For efficient checking, we set the block factor of all pinning sites equal to infinity (the 'large value' mentioned in Section 3.8.3) when we effect the pinning. The required check is realized by testing if the blocking factor sampled at $\mathbf{y}$ is infinity. This infinity-interpolation trick is also used in boundary trimming to indicate the regions where trimming should take place.

3.10 Ink Painting Results

Figure 3.9 was painted by first drawing a circle with glue in the center to create a 'wall'. Then, we painted a few strokes with a brush loaded with an ink gradient inside the wall. Notice the unevenness of the wall allowed some of the ink to seep through near the top part of the wall. We then painted the rest of the painting with different ink concentrations. Figure 3.10 shows real branching patterns made with *ink catalyst* (a relatively new art material), and some digital ink marks mimicking that effect made with

our system. We model catalyst as a material that blocks the ink flow. The digital marks were made by first spraying some catalyst onto the paper, and then adding red and yellow inks and letting them flow. In Figure 3.11(a), glue was used to hold some parts of the painting stationary. All the above figures were painted with a simulation resolution of 512^2 and a *d-scale* of 3.

Figure 3.9: A sample painting created with our system (res.: 1536 x 1536).

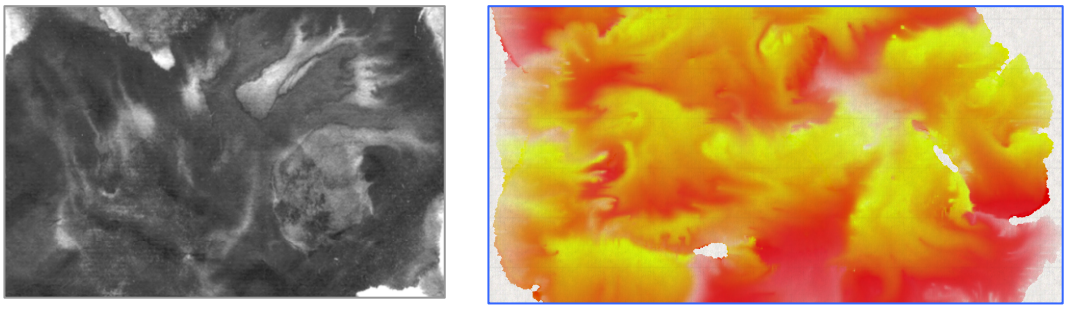


Figure 3.10: Severe branching patterns. Left: Real ink marks made with ink catalyst. Right: Marks of similar effect made with our system. No manual pushing or sucking of ink with brush was used.

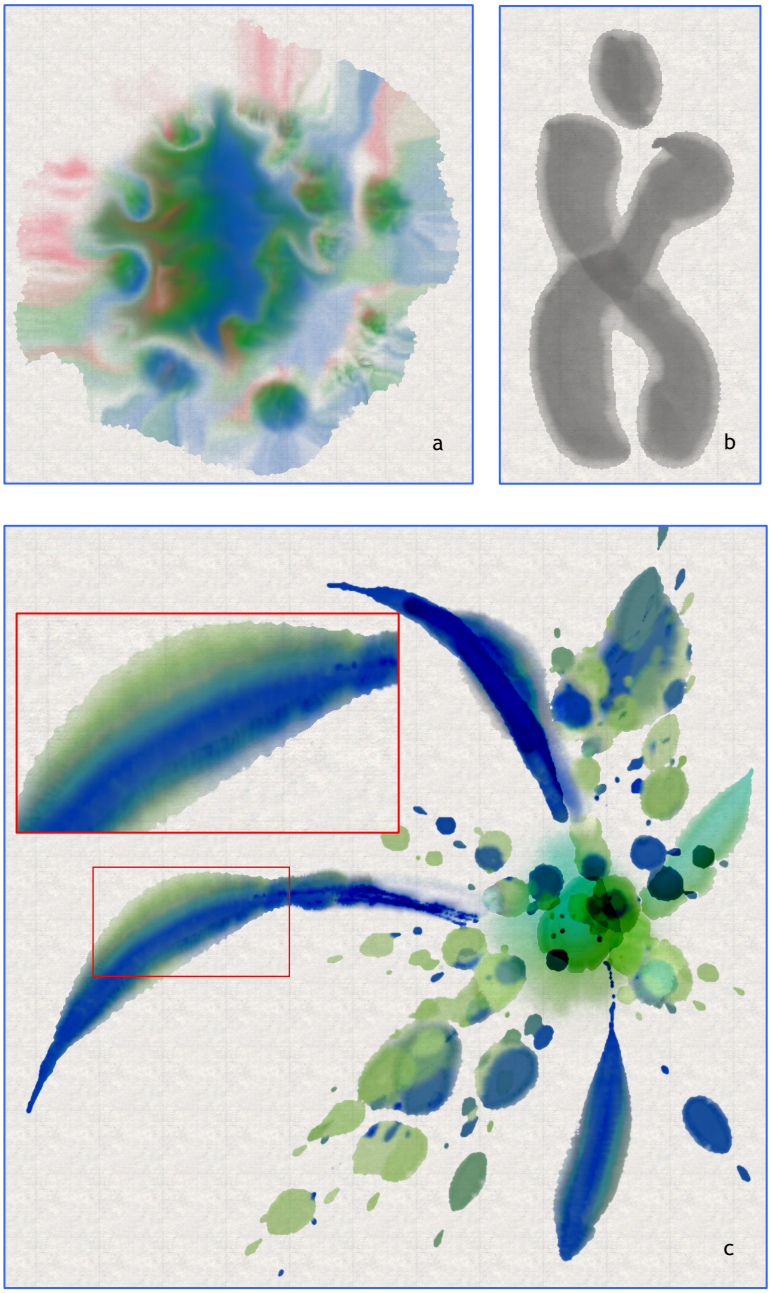


Figure 3.11: Sample paintings created with our system.

In Figure 3.12, Orge Yoko made use of water to displace ink to her advantage to draw the white strokes to outline the above-water-surface part of the bottle. Figure 3.13 shows an artwork commissioned by the public relation office of our university. This is a good example of using a split brush to make more visually pleasing strokes. The ink dispersion also played a part in creating the organic shape of the thicker strokes.

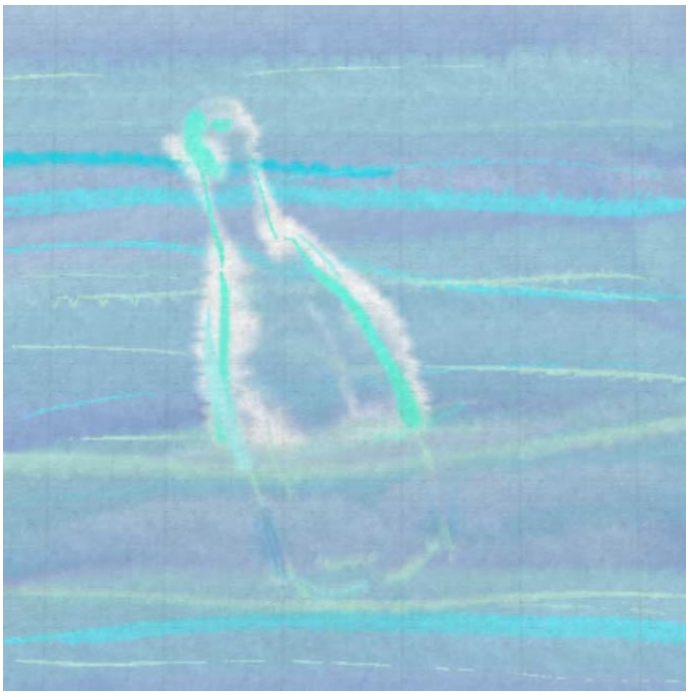


Figure 3.12: Sample painting by Orge Yoko (res.: 1024 x 1024).

Figure 3.13: Sample calligraphy made with MoXi. This work was vectorized using commercial software tool and was used as a Chinese New Year greeting card.

3.11 Special Flow Effects

In this section, I share my own experience in creating special effects useful for movie or interactive art. Our simulation has also been used by computer graphics studios around the world to produce special effects for movies or TV commercials.

3.11.1 Special Painting Effects

We have experimented with several special effects or controls that are hard or impossible to get in real life:

- Rewetting and moisture control: We provide the option to transfer pigments in the fixture layer back to the flow layer, so that artists can rework dried ink marks. We also add the controls of moisture redoubling (Figure 3.14(a)) and quick drying. Repeated addition and reduction of moisture can also create certain ring patterns.

- Physics tinkering: By switching the paper parameters suddenly during painting, we are able to produce some interesting patterns like Figure 3.14(b). By making certain (unphysical) modifications to our pigment advection scheme, we are also able to obtain nice flow patterns like Figure 3.14(c).

- Splash and spray: We also incorporated simple physics for ink drops (Figure 3.11(c)) so that artists can splash or spray art materials as in real life or have the ink drops emitting in some exotic patterns.

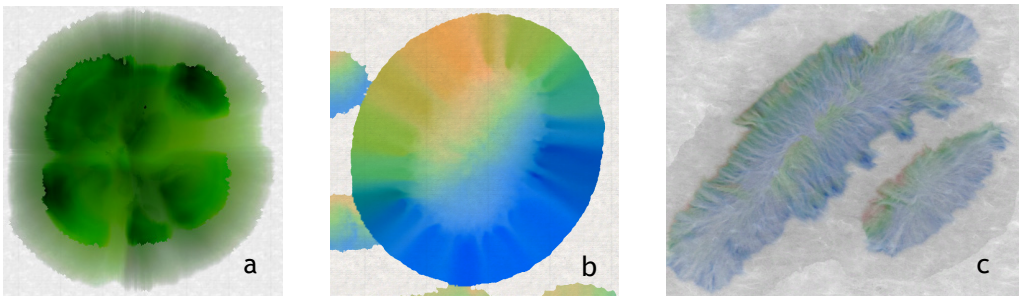


Figure 3.14. Left: Sample special ink effects. (a) moisture redoubling. (b) shaft-like patterns. (c) stream-like patterns.

3.11.2 Flare and Wash Animation

The flare and wash effects was specifically devised for use in a movie. The film director asked for an effect to express an angry feel for a scene's backdrop in support for the foreground action. The backdrop was to be some Chinese ink painting and I was asked to give an effect on the background to convey that angry feel.

To produce the flare effect, I first threshold the original image to obtain an ink source map (Figure 3.15). I use this ink source image to specify to my ink simulation the part of the image acting as a water source right on the 2D paper space. This makes the ink to flow outward, but this alone would only give ordinary-looking flow. To produce the flare-like flows, I also use a custom block map with big blobs as shown in Figure 3.15 right. Those blobs obstruct the flow creating the desired flare (Figure 3.16 middle).

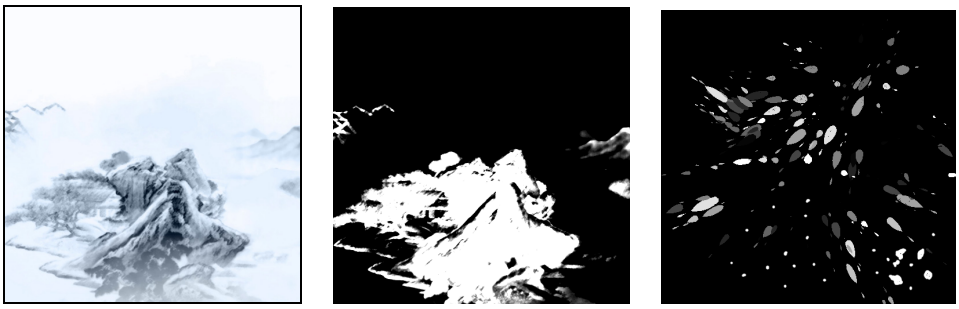


Figure 3.15. Left: Original input painting. Middle: Ink source map. Right: Block map. (Image credit: ink painting image from Menfond Eletronic Arts)

Figure 3.16. Snapshots of 'flare' and 'wash' effect animation.

The wash effect (Figure 3.16 right) was simply done by having the top of the paper acting as a water source. This is done by rendering a one-fragment wide rectangle at the

paper top with fragment program that sets the desired water density. Water velocity can also be set to point downward to accelerate the flow. Technically, the specified water density and velocity are referred to as *boundary conditions* in fluid simulation literature.

3.12 Simulating Western Watercolor

As mentioned in Section 3.1.5, there are several differences between Eastern ink and Western watercolor. Here I describe how I model *granulation*, *backrun*, *glazing* unique to watercolor.

3.12.1 Granulation

This results from the pigments settling at the valleys of the paper, giving a grainy texture. In the MoXi system, there are three layers representing the paper: the surface layer, the flow layer and the fixture layer. The surface layer acts as a supply of ink to the flow layer, where the fluid simulation of water and advection of pigment particles by the water is carried out. As the ink dries, pigments are transferred from the flow layer to the fixture layer. In the simulation of watercolor, we will use the flow layer to simulate flowing water on the paper surface. Granulation is implemented by modulating the rate of pigment transfer from the flow layer to the fixture layer by the paper grain texture. The flow speed also controls this transfer rate to create more variation in the resulting pattern. Essentially, granulation is implemented by the following Cg code:

```
if (length(u) < SettlingSpeed)
  FixBase += (1-smoothstep(0, GranulThres, grain)) * Granularity;
```

where *u* is the water velocity, *FixBase* is a base fixture rate, *grain* is the grain texture value. User specified parameters *SettlingSpeed*, *Granularity*, *GranulThres* controls the effects.

3.12.2 Backrun

Backruns appear when a puddle of water spread back into a damp region. The water tends to push the pigments creation a complex branching pattern, and often comes with sever darkened edges. My implementation of the backrun effect is basically a combination of edge darkening effect and desorptions of pigment already deposited back to the puddle of water. In the original MoXi system, we block the flow of water when

there is not enough water pressure for a wet region to flow into a dry region. This blocking is referred to as *pinning* (Section 3.7.3). The edge darkening effect is realized by lowering the water amount at pinned edges, which induces a flow of water, together with pigments, toward the edges. When we implement backrun, the only difference is that we now have to check for two pinning situations instead of one: the interface between dry and wet regions, and the interface between regions with water on the paper surface and those that are drying but still damp. Specifically, we now have three paper region types: *dry*, *damp*, and *wet* (Figure 3.17):

- Dry: without any water
- Damp: no water on paper surface, but there is some in the fiber of the paper
- Wet: there is water on the paper surface

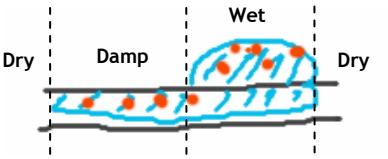


Figure 3.17: Illustration of dry, damp, and wet regions. Blue strokes indicate water. Orange dots indicate pigment particles.

I use the fact that real pigments in the damp regions are almost settled to simplify the simulation. I simulate the flow of water on the surface in the flow layer. Water is removed from the flow layer due to evaporation until no more left. When a lattice site just becomes dry, I set up a counter to mark that site as a damp site. This counter acts as a time-to-live counter and is incremented each frame. When it reaches a predefined value, the site is reset to a dry site. With the help of this counter, I can perform two pinning operations, one between dry and wet/damp region, and the other between dry and damp regions. In Cg code, the desorption of pigment mentioned above is implemented by moving pigments in the fixture layer back to the flow layer like this:

```
if (length(u) > SettlingSpeed)
  SinkInk = - FixInk * UnFix * wf;
```

where *SinkInk* is the amount of pigment to be transferred from the flow layer to the fixture layer (negative value means desorption), *FixInk* is the amount of pigment already in the fixture layer, *UnFix* is a user-specified parameter for controlling the amount of desorption, and *wf* is the amount of water in the flow layer.

3.12.3 Glazing

Glazing is the process of adding very thin strokes of watercolor paint, one over another, to create a clear and even effect. Curtis *et al.* [1997] claimed that in glazing, pigments are blended optically rather then physically. Following Curtis, I also implemented glazing using the Kubulka-Munk (KM) model [Curtis *et al.* 1997], which is reviewed in this section. However, more research leads to my discussion on the appropriateness of KM model on watercolor, which I present in Section 4.2.

In the KM model, each pigment is assigned a set of absorption coefficients K and scattering coefficients S. These coefficients are a function of wavelength, and control the fraction of energy absorbed and scattered back, respectively, per unit distance in the layer of pigment. Similar to [Curtis *et al.* 1997], I use three coefficients each for K and S, representing RGB components of each quantity. Assuming we already have set of K and S parameters for each pigment layer, the KM model computes the reflectance R and the transmittance T through the layer as:

$$R = \sinh bSx / c \qquad T = b / c$$

where $c = a \sinh bSx + b \cosh bSx$, and x is the thickness of the layer. For blending multiple layer one on top of another, we use the formula:

$$R = R_1 + \frac{T_1^2 R_2}{1 - R_1 R_2} \qquad T = \frac{T_1 T_2}{1 - R_1 R_2}$$

where the overall reflectance R and transmittance T of two abutting layers with reflectances R_1, R_2 and transmittances T_1, T_2, respectively, with R_1 and T_1 referring to the top layer.

For specifying the coefficients K and S of various pigments, I implemented the inversion of the KM equation, just like what Curtis *et al.* did. Given these two user-selected RGB colors R_w and R_b, respectively, the values of K and S are derived by:

$$S = \frac{1}{b} \cdot arc\coth\left(\frac{b^2 - (a - R_w)(a - 1)}{b(1 - R_w)}\right)$$
$$K = S(a - 1)$$

where

$$a = \frac{1}{2}\left(R_w + \frac{R_b - R_w + 1}{R_b}\right), \quad b = \sqrt{a^2 - 1}.$$

Using the above formula, however, we found that it's not possible to specify all colors, for example, pure red with RGB = (1, 0, 0), as derived values could be NAN or infinity. Therefore, we used three pigment colors close to yellow, magenta, and cyan (CMY), as our base pigments so as to obtain a large gamut of color by mixing the three base colors.

3.12.4 Results

In this section, I demonstrate the effectiveness of our paint flow model by showing organic strokes made with our system. Figure 3.18 shows our watercolor system running in watercolor mode in general. Figure 3.19 shows some close-ups of flow effects. The flow pattern in Figure 3.19 left was with a brush loaded with color and water. In Figure 3.19 left, the flow pattern was created by first laying down a yellow stroke and then an orange stroke. The orange stroke was extended to touch the yellow stroke and paint creep into the yellow stroke because there was more water in the orange stroke.

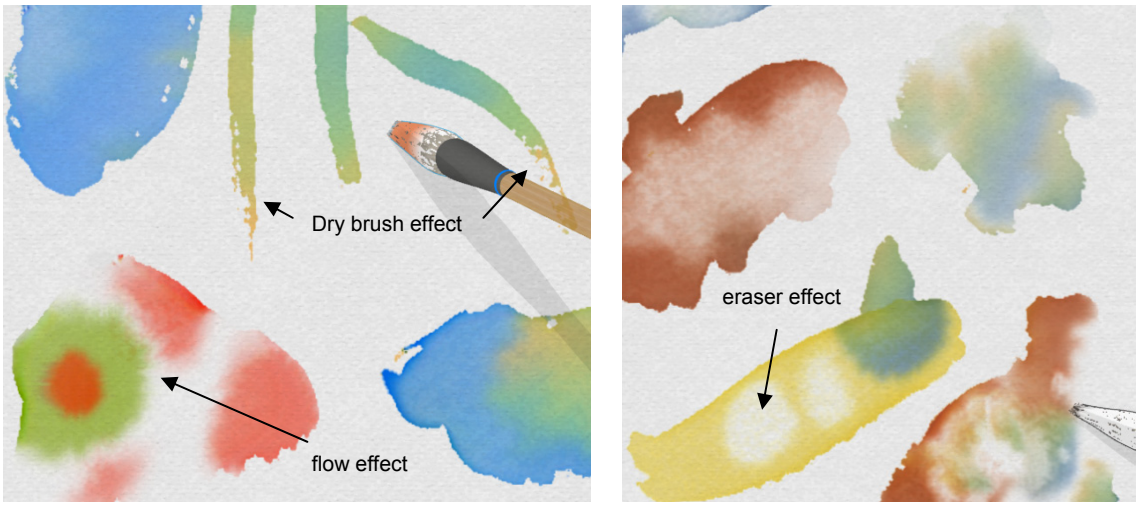


Figure 3.18: Screenshots of watercolor simulation.

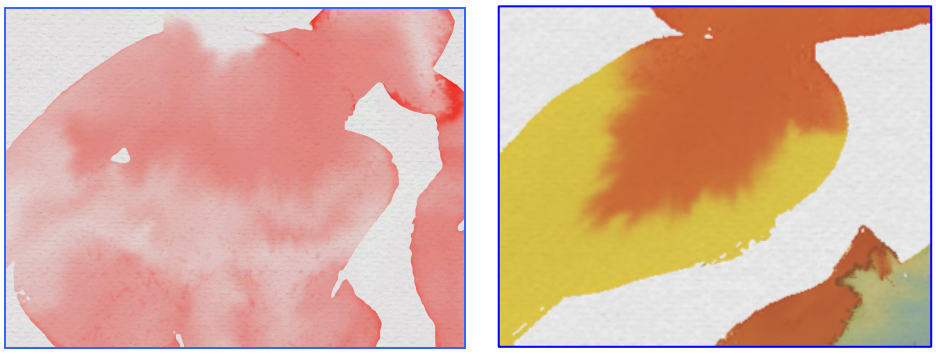


Figure 3.19: Sample watercolor flow effects.

When a stroke is still wet, depositing pure water on it would produce backruns or *oozles* like those shown in Figure 3.20 (a) and (b). Notice that there is not sharp edge around the oozles. This was produced not by the pinning described in Section 3.12.2, as the *damp region* was not yet formed. When *damp region* is formed, the oozles would have hard edges as shown in Figure 3.20 (c). It is also possible to obtain both wet-in-wet and backruns in one single stroke depending on the wetness on the paper (Figure 3.20 (d)). In conclusion, we are able to produce a range of effects from *diffusive flow* to *backruns with soft edges* to *backruns with hard edges* depending on the paper wetness just like in real watercolor (Figure 3.4 left).

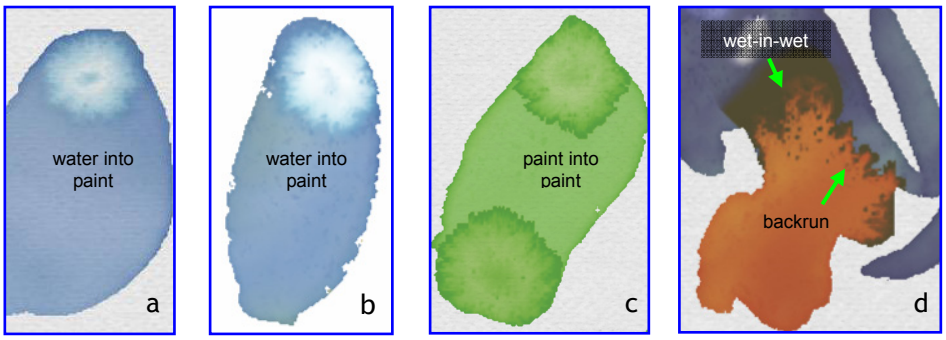


Figure 3.20: Wet-in-wet and backruns effects.

Figure 3.21 shows some granulation effects. The left image shows granulation produced only with texture modulation. The middle and the right ones show effects of granulation with the grain texture more influential to the water flow. Figure 3.22 shows strokes optically blended using the K-M model. There are two layers: the bottom one contains the horizontal strokes while the top one the vertical strokes. Notice that when the two layers are merged into one without any optical blending, the rendered result looks flatter (close-ups in Figure 3.22).

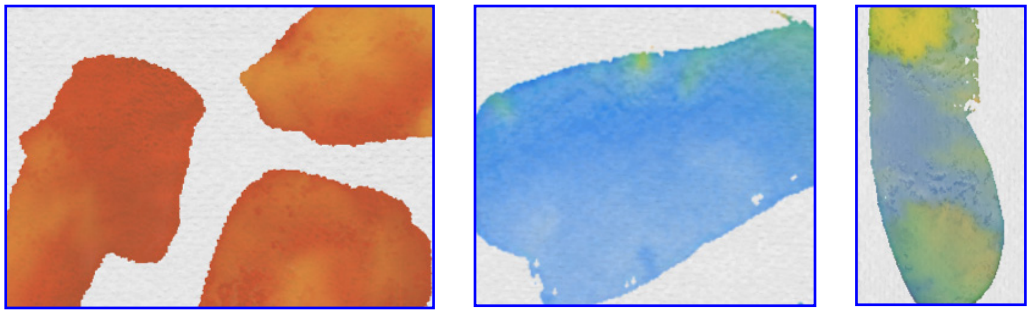


Figure 3.21: Sample granulation results.

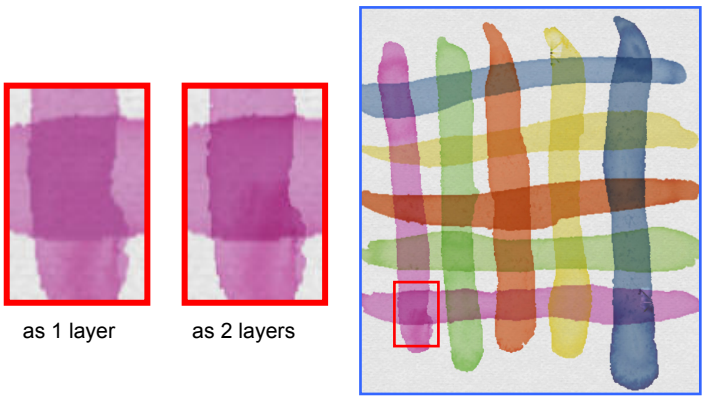


Figure 3.22: Glazing results rendered using the KM model.

3.12.5 Discussion

The current simulation of granulation is only an approximation and it does not always give a realistic or pleasing (in my opinion) effect. A more realistic approach is to simulate the movement of pigment particles within the water. However, I believe realistic granulation can be cost-effectively simulated with image-based approaches like texture synthesis [Lefebvre & Hoppe 2005; 2006]. Another effect that could be better implemented is the edge-darkening effect - using a non-physical approach like the edge-detection method [Curtis 1999] used in the animation *Fishing* [Gainey *et al.* 1999], the darkening can be adjusted without re-running the whole simulation, which is valuable for artistic control. Curtis *et al.* [1997] mentioned that it would be a significant improvement if one can simulate backrun and wet-in-wet flow as two extremes of a continuum effect depending on the paper wetness. Although my currently backrun simulation is approximate, I believe visually we could already give a full range of effects between flow and backrun as shown in Figures 3.19 & 3.20.

3.13 Simulating Suminagashi

In this section, I present the adaptation of our original ink flow model to simulate the ancient art of 'suminagashi' or floating ink on water surface. I will first give an introduction of the art form followed by a summary of previous work. Then, I will present the technical details of the simulation.

3.13.1 Suminagashi the art form

'Suminagashi' refers to the ancient art of floating ink on water surface and then capturing the flow pattern on a piece of absorbent paper. It is believed to be originated from China some two thousand years ago and spread to Japan in the 12th century. 'Suminagashi' or '墨流し' is a Japanese term meaning 'ink flow'. Today in the Chinese art community, most people can the outcome 水拓畫 meaning 'prints from water' in Chinese. Because it is already widely accepted and that it is easier for Western people to pronounce, I will keep using the term 'suminagashi' in this thesis.

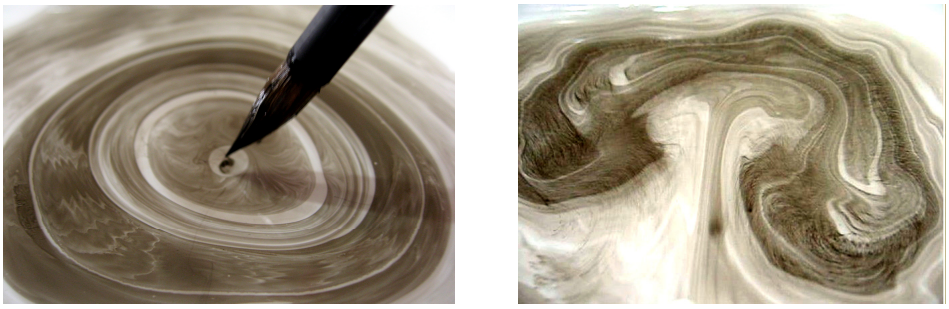


Figure 3.23. Left: Depositing ink in the art form of suminagashi. Right: Blowing air to the water surface to form flow pattern.

Suminagashi is believed to be the oldest form of *marbling* [Maurer-Mathison 1999]. Marbling refers to a method of producing decorative design by flowing paint on a liquid surface. The resultant patterns are similar to marble or other stone, hence the name.

In the West, marbling usually refers to *ebru*, the Turkish way of marbling. The main different between ebru and suminagashi is the viscosity of the fluid used. In ebru, the water is thickened with *size* or *fixative* so that any disturbance of the fluid is mostly local, so that the artists can have more precise control. Creating figures like flowers is feasible. In contrast, water in suminagashi is not thickened and so the flow is more fluid and it is actually hard to control the resulting shape. For a detailed discussion of marbling method, history, and sample results, readers are referred to the Wikipedia [Wikipedia Marbling 2007].

It is interesting to observe the different in philosophy in Eastern and Western art as Anne Chambers [1991] noted:

> *"In the West, the artist strives to master and control his materials in order to achieve a preconceived end. In the East, the materials must be allowed to do what they would do naturally; thus the sculptor takes from the piece of wood the form which the wood itself indicates. […] and the suminagashi artist invites his ink to go where they will on the surface of the water, with minimal assistance from the human agent. […] The suminagashi artist should not try to exercise control, but rather to relinquish control to a large degree."*

In this part of my work, I attempt to do my simulation in the Eastern way as I like the philosophy and the outcome looks more interesting to me. The more-controlled ebru is left as future work.

3.13.2 Previous Work

In the commercial software market, *Corel Painter* [Painter 2007] has a digital marbling feature since version 6 (then sold under the company *MetaCreations*) released in 1999. There are also a number of plug-in for making marbling for use with *Adobe Photoshop* or *Corel PaintShop Pro*. Since version 6 released in 2000, *Photoshop* also has a 'liquidfy' feature that allows distortion of an existing image in a way like dragging a brush through liquid. All these tools are commercial and no technical details are published. As far as I can tell, the only commercial product that uses fluid dynamics simulation in making fluid like effect is *KPT Fluid* in *Kai's Power Tool 7* released in 2001 by *MetaCreations*. KPT Fluid seems to be based on Stam's paper published in 1999 [Stam 1999]. Performance-wise, all the above tools run in real-time. It seems KPT Fluid's real-time performance is due to the simulation being limited to 256 × 256 or lower. In my opinion, the above non-physically based simulation of marbling with thick liquid is already very successful in terms of resulting image quality. It is also very practical since it does not require expensive computation as needed in physically-based simulation.

In the literature, the earliest published work on digital marbling is by Mao *et al.* [2003]. Mao's simulation is based on Stam's fast Navier-Stokes fluid solver [1999]. Although fast solver of Stam is used, one time step still required half a minute at a resolution of 512 × 512 on a Pentium III PC so interactive use is prohibitive. One of the main differences bewteen Stam's simulation of dye in a fluid and that of marbling is that marbling requires the ability to have sharp paint interfaces. Digital simulation is well-known to have numerical diffusion [Wikipedia Numerical Diffusion 2007] and this would cause blurriness in the resulting images and this is actually one of the challenges in recreating the sharp interface. Stam's semi-Lagrangian solver is known to give large dissipations in both dye amount and fluid momentum if the time step is set large in compromise for real-time speed [Song *et al.* 2005].

Yoshida *et al.* [2004] performed a suminagashi simulation using a haptic input device. In their work, instead of using physically-based fluid dynamics, simple smearing or smudging akin to those found in standard image-editing software tools was used. Because of the smudging, the resultant image can be quite blurry.

In the work of Acar and Boulanger [2006], dynamics of the marbling fluid is solved using a mescoscopic approach. Paint pigments advection is simulated with hydrodynamic dispersion. For obtaining sharp interface between regions of different paint colors, they use a transfer function to map graduate variations in the paint concentration to sharp transitions. Very nice results are obtained but the simulation is not likely to be real-time (no performance data is given). Another approach for sharpening interface based on level-set is used in [Weiskopf 2004]. This level-set based method is computational expensive due to the level-set reinitialization.

Jin *et al.* [2007] updated Mao's work with GPU implementation running at 24 FPS with simulation resolution of 512 × 512 on an nvidia Geforce 7800 GS. According to Jin *et al.*, the smooth transitions resulting from texture interpolation is the main cause of blurriness in their algorithm. To alleviate the blurring problem, they sharpen the resultant paint flow patterns with *shock filtering* for display purpose.

3.13.3 Simulation Overview

Much of the processing originally in the ink flow simulation of MoXi can be removed for suminagashi for speedup. This includes the render-passes of StainMap, the transition of ink from flow layer to fixture layer, the boundary trimming described in Section 4.1.1.

Technically, suminagashi simulation is the same an ink flow simulation except with the following special treatments:

1. Flow impedance set to almost zero;
2. Ink pigment advection accelerated;
3. Ink color re-mapped with a texture map.

The first point is easy to do. The other two points needs some further elaboration.

3.13.4 Pigment Advection Acceleration

This is done with a very simply modification to the pigment advection render-pass. Recall that in the method of characteristics, new pigment concentration is obtained by

$$\mathbf{p}_f^*(\mathbf{x}) = \mathbf{p}_f(\mathbf{y})$$

where $\mathbf{y} = \mathbf{x} - \mathbf{u}(\mathbf{x})$. Now, my change is to add a scaling factor a to the tracing vector so that $\mathbf{y} = \mathbf{x} - a\,\mathbf{u}(\mathbf{x})$. The value a can is usually 10 for normal suminagashi-like simulation. For more dynamic effects, one can even set a to great than 100.

3.13.5 Color Remapping

This step is to use the ink concentration to index a predefined 1D texture map and use this texture value as the output color. Let us denote this map by *ColorMap*. This in theory would allow much higher resolution rendering as now a small variation in color could be mapped to a wide range of values defined by the *ColorMap*. This is a very cheap way to obtain sharp ink streaks without using any complicated method for keeping ink boundary shape.

3.13.6 Results

On a Geforce 7800GTX, when all the unnecessary rendering passes mentioned in Section 3.10.3 are removed, my suminagashi-like simulation runs at 25 fps when water flow is simulated at 1024 × 512, ink map advection at 2048 × 1024, and rendering output at 1920 × 1080.

Figure 3.24. Visual effects made with MoXi suminagashi running on SONY's Cell Computing Board.

MoXi has also been ported to the upcoming *Cell Computing Board* from *SONY Electronics Inc.* (Figure 3.24). Like *PlayStation 3*, this Cell computing board is equipped with a *Cell BE* processor and a GPU called the *RSX* (Reality Synthesizer). The RSX is a GPU based on an nvidia G70 chip. I implemented shaders classes and control logic in C++, while the whole fluid simulation and ink rendering are done on the RSX. The Cell platform supports a lite version of OpenGL and the nvidia Cg 1.5 so my port of the fluid simulation was relatively easy. The resulting application was demonstrated in SIGGRAPH 2007. Some samples artwork made with MoXi suminagashi are shown in Figure 3.25.

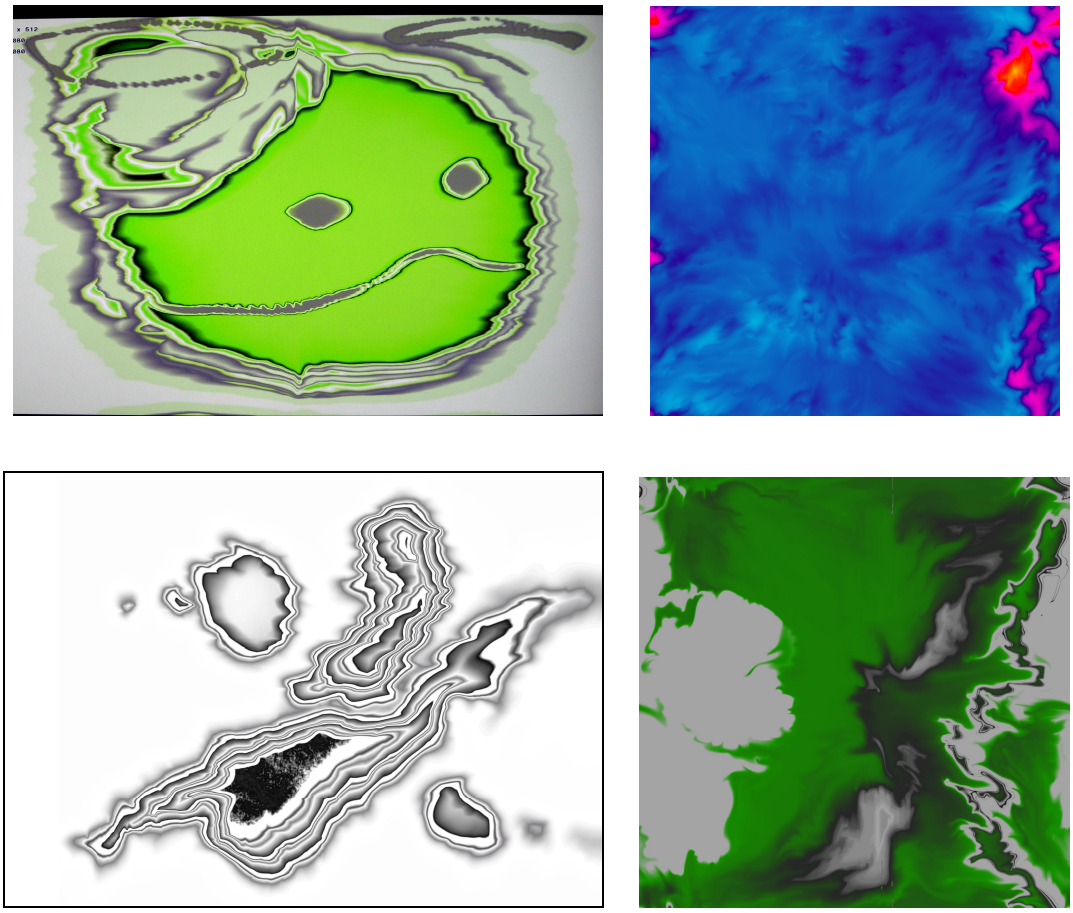


Figure 3.25. Sample MoXi suminagashi artwork.

3.14 A note on Paint Advection and Stability

The LBM is stable given water density and velocity within the designated ranges. However, I want to be able to push ink around and my first attempt was to do this by setting water velocity in the lattice sites touched by the brush. The problem with this approach was that the speed of ink movement is also often too slow to match that of the brush (brush can move at a speed of tens of lattice spacing per time step while ink advection at most one lattice spacing per time step). I attempted to allow faster movement by setting large values in the water velocity but the assumption of small variation in the water density field in the LBM is violated leading to simulation blow-up.

The multiple-relaxation-time (MRT) formulation of the lattice Boltzmann method described in [Lallemand & Luo 2003] allows a more stable simulation and thus I wanted to implement it to allow ink to move faster during manual pushing. In fact, the MRT formulation has already been used by Qiu *et al.* [2004] in their dispersion simulation. However, I found that the implementation of MRT would increase my simulation time quite a lot: apart from the extra calculations needed for transforming values in different spaces, the relaxation done in the moment space does not fit nicely into the four-vector (RGBA) parallelism of the GPU. This probably would break the real-time quality of my system, which is very undesirable.

So, for avoiding simulation blow-up, I simple clamp the values that could cause the blow-up. For example, I clamp the water density to make it without the valid range of [0..1]. I have found this simple low-cost solution working rather effectively.

Nevertheless, the need for fast paint advection to match interactive user input is still not satisfied by the above trick. Finally, I solve the problem by the 'accelerated paint advection' method described in Section 3.13.4. This method is analogous to the idea of semi-Lagrange method [Stam 1999] for accelerating fluid flow simulation, in which the back-tracing vector is scaled larger than they should be to allow accelerated simulation. When used alone (i.e. without other treatments for suminagashi simulation), I call the accelerated paint advection i*nk-rush*. A sample image produced with ink-rush is shown in Figure 3.26.

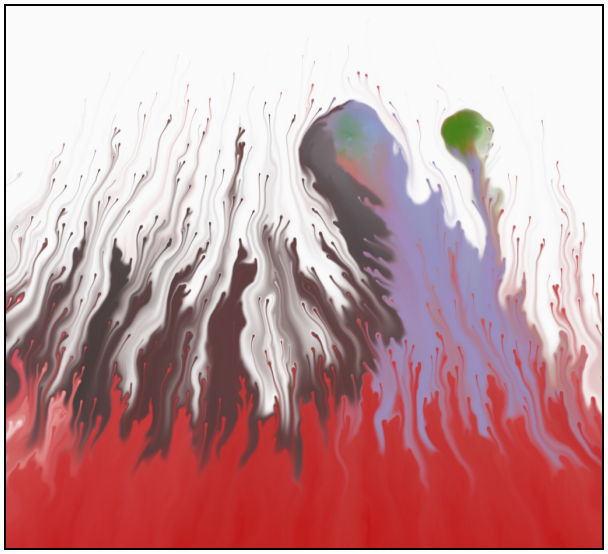


Figure 3.26: Sample ink-rush effect made with advection scaling factor *a* set to 80.

CHAPTER 4

RENDERING

In the first part of this Chapter, I describe a few tricks that I use to improve the rendering of brush marks. In the second part, I discuss the use of the Kubulka-Munk model, which is very commonly used among computer graphics researchers.

4.1 Resolution Enhancements

It is important that a paint system can render high resolution output since artworks are often produced at print resolution. However, doing a physical simulation down to the scale of tiny paper fibers for fine details would be prohibitive. Besides, excessive simulation is undesirable if similar results can be obtained at a much lower cost. Therefore, we enhance the resolution of our ink simulation output with implicit modeling and image-based methods. We choose to keep our methods simple and fast so as to give immediate user feedback; this contributes towards a better painting experience.

4.1.1 Boundary Sharpening

To generate a higher resolution output, we first take the original simulation output I and produce a larger version I' by hardware interpolation. The interpolation blurs away sharp features and the boundaries look blocky when enlarged (Figure 4.1(a)). To keep the boundaries sharp and avoid blockiness, we trim away pixels that are interpolated between stained and unstained sites in I'. Conceptually, the trimming scheme is similar to iso-surface methods [Whitaker 2002].

To trim the ink marks in I' on the fly, we must first decide when and where to trim. Basically, the life of an ink mark has three stages: (1) expanding as the paper imbibes it, (2) having its boundary pinned, and (3) having its wet area shrinking as the water evaporates. Clearly, the trimming should only be done during the second stage. When

this condition is detected (by keeping track of some site variables), trimming is performed according to an implicit curve defined by a scalar function ϕ derived from the water density field. Each lattice site is associated with a variable ρ_τ, which is updated at each time step to $max(\rho_\tau, \rho)$ if the site is wet, and to a small negative value if it is dry. The function ϕ interpolates the values of ρ_τ in different sites. We use $max(\rho_\tau, \rho)$ instead of ρ so that ρ_τ would not change as the mark dries (during its second life stage). We first shrink the implicit shape a bit by overwriting ρ_τ at the sites just inside the boundary with zero, and then trim the pixels of I' where $\phi < 0$ to give a sharp and natural-looking boundary.

For fine boundary roughening that is not captured by the coarser resolution of I, we also add a roughening term when deriving ϕ:

$$\phi = \rho_\tau + r_{scale}(r + r_{bias}) \ , \tag{11}$$

where r_{scale} and r_{bias} are parameters for adjusting the effect, and r is the value of a *rougher texture,* which is simply a blurred image of some random spots. To generate smooth boundaries, we use $r_{scale} = 0$ and replace the term ρ_τ in Eq. (11) by the value sampled at the same point from a low-pass-filtered $[\rho_\tau]$,with $[\rho_\tau]$ denoting the texture storing the data field ρ_τ. Figure 4.1(b) shows a trimming result.

In practice, the texture storing the pigment concentrations in fixture, denoted $[\mathbf{p}_x]$, is used as the input for generating I'. Ideally, I' should be updated at every time step to reflect the changes in $[\mathbf{p}_x]$. However, for better performance and to avoid a hardware precision issue (described in Section 3.9.3), we only update I' every 200-400 frames. For rendering the painting scene, we also apply boundary trimming on $[\mathbf{p}_f]$ and $[\mathbf{p}_x]$ (every frame) so that the user sees nice boundaries while painting.

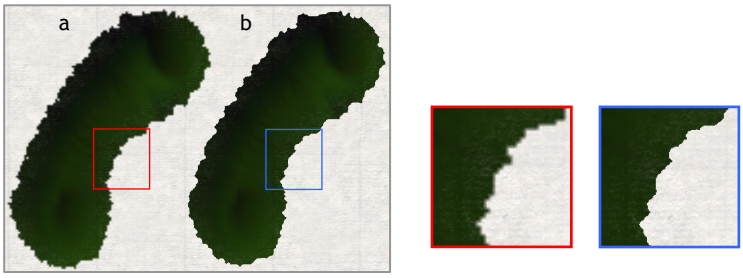


Figure 4.1: Boundary trimming. (a) Before trimming. (b) After trimming.

4.1.2 Anti-aliasing

When the above boundary sharpening is applied, *aliasing* [Wikipedia anti-aliasing 2007] occurs because the thresholding of ϕ described above simply does a sharp cut. The final bitmap output also bears 'staircase' artifacts (Figure 4.2, a=0) which is less than satisfactory. To relief this situation, I use the function *smoothstep()* instead of using a sharp step function for thresholding ϕ as described in the last section. Specifically, my simply anti-aliasing is done by multiplying the ink concentration I' with the pseudo code:

$$I' \leftarrow I' \times \text{smoothstep}(-a, a, \phi - \text{EdgeThres})$$

where *EdgeThres* is the thresholding value, and $a \in [0, 1]$ is a parameter for anti-aliasing. Because I' is already linearly interpolation by hardware texture filtering, multiplying it with *smoothstep()* gives smooth transition. Figure 4.2 shows the anti-aliasing effects with different values of a.

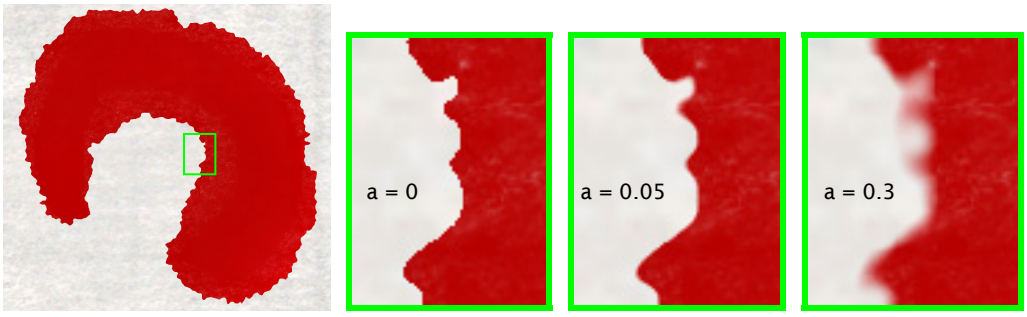


Figure 4.2: Simple anti-aliasing with different parameter values.

4.1.3 Textural Details

When ink percolates paper, tiny hairs of the paper fibers become apparent in regions not fully soaked by the ink (as in the left part of Figure 3.6(c), and the lower left corner of Figure 3.6(d)). We render this effect by modulating the pigment concentrations in regions that are partially soaked with a *hair texture*, which is prepared from real images of stained paper. The modulation also depends on paper thickness to give venetian blind patterns (Figure 3.6(c)). Sample results are shown in Figures 3.6(d) and 4.1.

4.2 Paint Appearance Modeling

The Kubulka-Munk (KM) model [Kubulka & Munk 1931] was designed for modeling optical blending of layers of translucent materials. It is widely used in the paint, printing, and textile industries to determine diffuse colors due to subsurface scattering [Judd & Wyszecki 1975]. The KM model was introduced to the computer graphics community by Haase and Meyer [1992] to render thick layer of paint consisting of several pigments. Since then, this model became popular in the graphics community for paint rendering for media like watercolor [Curtis *et al.* 1997; Lum & Ma 2001; Lei & Chang 2004; Laerhoven & Reeth 2005], oil or acrylic [Baxter 2004], and wax crayon [Rudolf *et al.* 2003]. With the popularity of programmable graphics hardware, the KM model has also been implemented on the GPU [Baxter 2004].

4.2.1 My experience in using KM Model

As mentioned in Section 3.12.3, I have also implemented the KM model on the GPU to render paint. Sometimes when I switch from 'cyan, magenta and yellow' (CMY to KM rendering, the picture suddenly looks more seasoned or interesting. Why is it so? In my opinion, it's the non-linearity which makes the paint look more subtle. Another reason is that the palettes represented by the KM do not comprise of standard RGB triples (like (1, 0.5, 0.5), (0, 0.5, 1)). Standard RGB colors give us a distinctive 'digital' look and we often get bored by them as most existing image editors provide such a palette by default.

However, I also realize that, as an artist, I do not always prefer using the KM model over the simpler CMY model. The KM model certainly models paint more realistically, but to me, KM and CMY or RGB are just *some* style of mixing colors, without one necessarily look better than another (Figure 4.2). As a painter, I also think that it is

sometimes useful to have a linear interpolation for color mixing as the resulting color can be more predictable. Whether the KM model is more 'accurate' does not really matter.

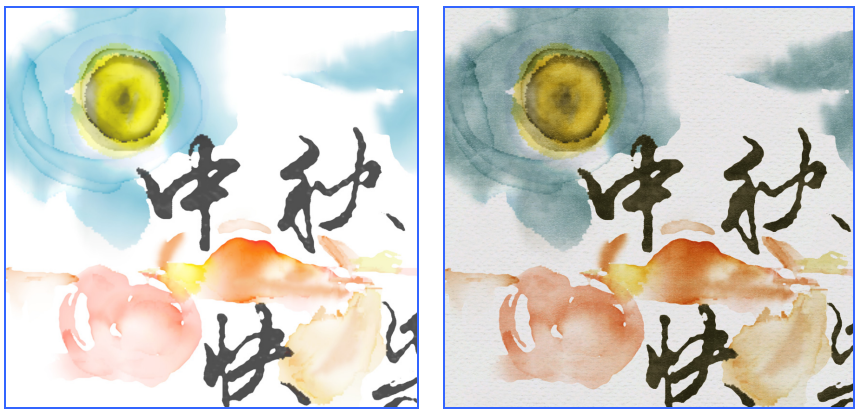


Figure 4.3: Comparison of RGB (left) and KM (right) color model rendered by MoXi.

4.2.2 Real Watercolor Mixing

Finally, although the KM is so popular in the computer graphics community, I would like to point out that real watercolor mixing defers from what the KM is designed for. In reality watercolor paints do not form paint layers [Handprint Watercolor 2007]. When watercolor paint is deposited on paper, the paint vehicle dries leaving pigment particles in between the tiny spaces between paper fibers (Figure 4.4). So, from a theoretically point of view, the KM is not a right model for watercolor since watercolor paint do not form layers of translucent materials. In fact, when they first introduced the KM model to computer graphics, Haase and Meyer [1992] pointed out that a subtractive color mixing is appropriate for watercolor since the paint is usually transparent enough to allow light to pass to the substrate (usually paper) and reflect back through the paint [Evans 1948]. The KM model is needed only for pigmented materials such as artist' oil paints.

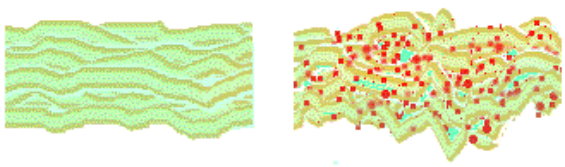


Figure 4.4: Microscopic reality of watercolor paint pigments on paper. Left: paper fibers before painting. Right: after paint dried (image from www.handprint.com © 2007 Bruce MacEvoy).

CHAPTER 5

USER INTERFACE

User interface affects greatly the user experience in using any computer program. In fact, one reason why some people do not like using computer is steep learning curve or hard-to-use interface. In this chapter, I first present a few ways to better visualize the virtual brush. Then, I describe my experience in making a 3D input device and discuss other input device alternatives.

5.1 Tuft Visualization

Visual feedback of the brush shape is important during the painting process. Traditionally, user interface of paint system is typically like that of a traditional image-editing tool: with menus on top, tool palette, color palette sitting on the sides with the 2D canvas in the middle.

In our system, a user can choose either an orthogonal or a perspective of the 3D painting scene. Brush shadow is rendered to enable the user to judge the position of the virtual brush. A common way to suggest the shape of the tuft is to shade it. However, some users may prefer to see the ink color loaded on the tuft unmodulated. In this case, we outline the tuft for legibility (Figure 5.1). The contrasting outline also avoids the tuft being ‘camouflaged’ when it is held over a ground of the same color as the ink loading, which is often the case. Details are presented in the Appendix. Finally, the brush can also be rendered transparent so as not to block the view to the painting.

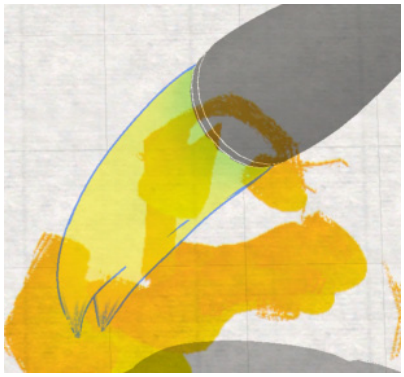


Figure 5.1: Tuft outlined for legibility.

5.2 Input Devices

Painters use all six degrees-of-freedom (DOF) 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. It is interesting to note that current commercial painting programs take little advantage of these extra degrees of freedom. Although my focus in this thesis is software technology, I also attempted to provide a good solution for input device to use with my paint system.

I bought my first graphics tablet back in 1993. One of the reasons why I thought I could not create digital brush marks just like I can in real life is the pressure sensing of graphics tablets. Pressure is sensed by a retracting tip of the stylus. At that time, I thought the retraction of 1 or at most 2 mm was just not good enough for 3D brush movement. So, during the early years of my digital brush making investigation, I attempted to make a 6-DOF input device, in hope that it could give me better manipulation of my virtual brush.

5.2.1 My Try on 6-DOF tracking

To drive the virtual brush, I have built a 6-DOF input device by combining affordable sensor components. I use an ultrasonic device and miniature gyroscopes to detect the brush position and orientation respectively. These sensors are attached to a real brush or a brush-like object, which is manipulated by the user in real-time (Figure 5.2). The input

device can be calibrated to map a real supporting surface to the virtual one, so that the real surface gives some tangible feeling to the user when the brush is pressed down. At present, the gyroscopes are still too heavy to allow a burden-free manipulation.

I have also experimented with an affordable 6-DOF glove-like controller that is based on optical tracking technology. The main drawback of this option is its bulkiness that forbids us to delicately control the virtual brush with our fingers.

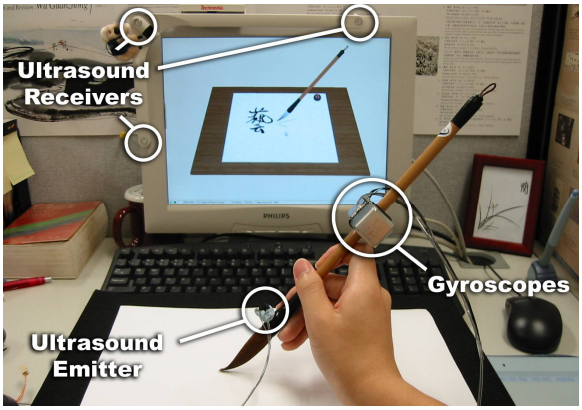


Figure 5.2: Using sensors to capture brush motion.

5.2.2 Tablets

As an alternative, our system also supports pressure and tilt sensitive graphics tablets (Figure 1.1). The sensed pressure controls the height of the brush above paper, while the tilt controls the orientation of the brush. The best tablet model commonly available today is a 5-DOF device which does not track the rotation about the stylus' own axis. Although in this case brush twisting is not sensed and the brush height is not controlled as directly as it would be with a true 3D positional device, the wide availability of tablet products makes our system more accessible to today's digital artists. The graphics tablet is also more convenient to use since a supporting ground already exists without calibration.

Currently, tablet is my primary input device. After using it for quite some time, I realized that the 1 mm pressure sensing is not really that bad as I thought.

5.2.3 Other Devices

Force-feedback input device are also available for quite some years already (Figure 5.3). In fact, it has been used in Baxter's own brush system [Baxter *et al.* 2001]. With these input devices the paint program can ensure that the haptic feedback presented to the user is consistent with the simulation state in the computer. Currently these haptic devices are still too expensive for most artists. Another issue is that an energetic user may find the robot-arm-like device actually restricting his or her arm movement.

Figure 5.3. A force-feedback input device (image from www.senable.com).

Multi-touch input devices [Han 2005] have become popular in recent years. When I was working on suminagahi simulation (Section 3.13), I think a multi-touch like the one developed by Han [2005] would be perfect for us to push ink around. I believe the combination of our software and such a device would make a very nice piece of interactive art work.

CHAPTER 6

CONCLUSIONS

I have presented techniques for simulation brush dynamics and water-based paint flow simulation for artistic creation purposes. I demonstrated the usefulness of such techniques by applying them on digital painting, animated visual effects and also interactive artistic creation.

The design of paint systems benefits greatly from artist's input. Being an artist and a software developer in one, it is easier for me to come up with a system that appeals to artists. Soon after we posted our SIGGRAPH 2005 paper preprint online, *Adobe Systems Inc.* approached us. Our MoXi brush and paint simulation technology is expected to be incorporated into the next version of *Photoshop*. The MoXi work has brought me the chance to work temporarily with big firms in the U.S. and Japan. Looking back, it was really a journey worth taking.

In the past few years, I have the chance to participate in various exhibitions, demonstrating MoXi to audience of very different backgrounds. Most audiences were supportive of my work, but I also met people commenting negatively saying that people doing real art just would not use such a digital replicate of the tools.

6.1 Hindsight

In the reply to reviewer comments from of one our paper submissions on brush modeling, I wrote:

> *Choosing to do sloppy physics would certainly speed things up considerably, but this comes with the price of losing the vitality of the brush, which defeats the purpose of developing such a digital brush in the first place. With the computation power keeps increasing for both CPU's and GPU's, we believe that we should take up the challenge.*

Fours years later, I still believe we chose the right path and I am glad to see more researchers are joining the quest for better brush models. It was my intuition to choose energy minimization over vectorial approach. There are other ways to simulate brush dynamics, but till now, it seems energy minimization is still the best choice for commodity PC application. Now, our same simulation running at 25 fps five year ago can run at 80 fps on a moderate PC. Yet, I still haven't taken advantage of the benefit of multi-core in modern CPU's. For paint flow modeling, I am also glad that I did not simply take Stam's scheme as my 'best' way to do real-time fluid (Stam made his code available, so it's really easy to use his code and be done with it; later Harris [2004b] also made his GPU code available). Thorough literature search was really worthwhile. I am glad that I found the Lattice Boltzmann Method (LBM) and I have to thank Wei *et al.* [2003] again for bringing LBM to the graphics community. The development of LBM is still unfolding and it is always exciting to browse through new papers titles from the physics community. Five years ago, I have not the faintest idea that GPU's would develop into a general-purpose processor as it is today and in such a short time. And, it is almost like the LBM is designed *specifically* for the parallel GPU's. All these developments came together at the right moment! Everything was like a gift from God to me.

6.2 Future Work

There are many things still need to be worked on. One major improvement would be on the Western watercolor medium but I believe a 'complete' simulation by itself probably worth one whole other thesis. It is also possible to improve the efficiency of brush simulation as described in Section 2.5.9. In the following sub-sections, I want to mention two major future work directions.

6.2.1 Rendering

Since the printing of large format digital artwork requires much higher resolution outputs, I am interested in further increasing the output resolution. Possible directions include texture synthesis techniques for fine details, and the use of Lagrangian particles [Sun 1996] in the simulation to resolve very fine streaks. Recently, the development of real-time texture synthesis on the GPU is very promising [Lefebvre & Hoppe 2005;

Lefebvre & Hoppe 2006]. The rendering can also benefit from the use of Wang tiles [Wei 2004] to give non-repeating paper textures.

6.2.2 New Effects

We should certainly continue to harness the programmability of computers to create new effects. Physicists have explored various fluid behaviors, including viscous fingering and surfactant physics, using the LBM. I am interested in exploiting LBM's advantage of easy addition of new fluid physics for new painting effects.

6.3 Closure

Perhaps it is also imperative to state the following. Digital art tools should not be meant to replace their real counterparts altogether, at least not in the foreseeable future. However, like different sports, the users have the freedom to choose which to pick up. As computers are becoming more and more popular, digital art tools may just become more and more popular.

In China, the brush has receded from its role as an essential tool for daily writing due to the invention of modern pen. In fact, even the pen is less utilized due to the use of computer. In making a digital Chinese brush, I also hope to lay down the foundation of further development of digital Chinese calligraphy. I am still not sure what I can bring to the table, but since I love calligraphy, I believe I should try my hands on further developing calligraphy on a computer.

APPENDIX

TUFT RENDERING

Our outlining method is based on [Raskar 2001], which applies well on polygonal meshes without texture transparency. Since our tufts have alpha map applied, we use additional render passes to make the alpha blended tip look nice. Here are the render passes of the modified method:

- **Pass 1:** Render back-faced lines with thick line width, outline color and texture alpha applied. Alpha value is thresholded (with alpha test or fragment program) to avoid the lines showing up in the alpha-blended parts (supposed to be covered by Pass 2) as artifact.
- **Pass 2:** Render back-faced polygons with outline color and texture alpha applied.
- **Pass 3 (optional):** Render front-faced polygons with outline color and texture alpha applied. Texture coordinates is slightly shifted towards the tip to make the splitting bristles more visible.
- **Pass 4:** Render front-faced polygons with normal color and texture alpha applied. Polygon depth offsets (*glPolygonOffset()* in OpenGL) should also be applied appropriately between the passes to avoid z-fighting incurred by rendering the same geometry multiple times.

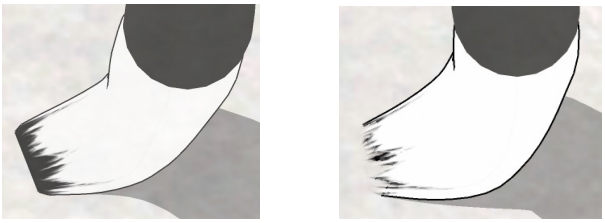


Figure A.1: Brush rendered with our method. Left: without taking care of texture alpha. Right: Texture alpha applied.

How we render multiple tufts with transparency

In general, fast and correct transparency for overlapping meshes is not trivial in OpenGL [Everitt 2001]. Here we use a little trick to render overlapping tufts as transparency. In our scene graph, the brush is the last object to be rendered. We render the tuft meshes as opaque at this point. Then, we set the paper surface transparent and draw it once more only on top of the fragments of rendered tufts with the help of the stencil buffer. The result is the tufts appearing to be transparent over the paper. This is possible because our 3D scene is a special case – the only important object behind the tufts is the paper and that the tufts are treated as a single object.

Using position as texture coordinates

The various deposition effects are actually performed in one single shader applied on the tuft object. These effects include paper height-field thresholding, which involves accessing the texture of the paper surface. Normally, a shader program applied on an object has no way to know the texture coordinates at an arbitrary point on another object. Fortunately, since the paper surface is just a 2D rectangle on the x-z plane, we can simply use the x-z positions of the tuft fragments as texture coordinates to the paper height-field texture. With modern GPU programmability, this can be done by just passing the vertex positions from the vertex program to the fragment program (with possible scaling and biasing), and these values would be interpolated among the fragments automatically by the GPU. The same trick can also be useful for implementing the intuitive color loading mentioned in Section 2.6. In this case, the tuft loading is stored as a texture on the tuft. This texture is to be updated by rendering the

tuft with the roles of x-z position and texture coordinates swapped: tuft geometry is flattened with its texture coordinates as the new vertex positions, and its x-z positions are used to index the texture of a color source. A fragment program checks if the original tuft object is touching the color source and determines which fragments of the tuft object are to take the old loading or are to be updated with new loading from the source. The resulting image is then used as the new tuft loading texture.

REFERENCES

ACAR, R. AND BOULANGER, P., 2006. Digital Marbling: A Multiscale Fluid Model. In *IEEE Trans. Visualization and Computer Graphics*, vol. 12, no. 4, 2006, pp. 600-614.

ADAMS, B., WICKE, M., DUTR´E, P., GROSS, M., PAULY, M., & TESCHNER, M., 2004. Interactive 3D painting on point-sampled objects. Eurographics Symposium on Point-Based Graphics.

ADLER, P., 1992. *Porous Media: geometry and transports*. Butterworth-Heinemann.

AKGUN, B.T. 2004. "The Digital Art of Marbled Paper," *Leonardo*, vol. 37, no. 1, 2004, pp. 49-51.

ALAVA, M., DUBÉ, M., AND ROST, M. 2004. Imbibition in disordered media, *Advances in Physics 53*, 83-175.

ARTRAGE 2007. Artrage 2.5 by Ambient Deisgn (www.ambientdesign.com).

BARRASS, G. S., 2002. The art of calligraphy in modern China. University of California Press, 2002.

BAXTER, W. V. SCHEIB, V., LIN, M. AND MANOCHA, D., 2001. DAB: Interactive haptic painting with 3D virtual brushes, SIGGRAPH 2001 Proceedings, ACM Press, 2001, pp. 461-468.

BAXTER, W. V. AND LIN, M. C. 2004. A versatile interactive 3D brush model. Proc. Of Pacific Graphics 2004, pages 319–328.

BAXTER, W. V., LIU, Y., AND LIN, M. C., 2004A. A viscous paint model for interactive applications. Computer Animation and Virtual Worlds, 15(3–4):433–442.

BAXTER, W. V., WENDT, J., AND LIN, M. C., 2004B. IMPaSTo: A realistic model for paint. In Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering, p 45–56.

BAXTER, W. V., 2004. Physically based interactive painting. PhD Thesis. University of North Carolina at Chapel Hill.

BOGHOSIAN, B. M. 2003. A Look at Lattice Boltzmann Equations. *Computing in Science and Engg.* 5, 2 (Mar. 2003), 86-87.

BREEN, D.E., HOUSE, D.H., AND WOZNY, M.J., 1994. Predicting the drape of woven cloth using interacting particles. Computer Graphics (Proc. SIGGRAPH), pages 365–372, 1994.

CHAWISK, J. E., HAUMANN, D. R., AND PARENT, R. E., 1989. Layered construction for deformable animated characters. *SIGGRAPH'89 Proceedings*, pp. 243-252, July 1989.

CHEN, S., DOOLEN, G. D., AND EGGERT, K. G., 1994. Lattice-Boltzmann Fluid Dynamics: a versatile tool for multiphase and other complicated flows. Los Alamos Science, Number 22, p. 98-111, 1994.

CHU, N. S. H, AND TAI, C.L., 2002. An efficient brush model for physically-based 3D painting, Proc. of Pacific Graphics'02, IEEE CS Press, 2002, pp. 413-421.

CHU, N. S. H. AND TAI, C.L., 2004. Real-Time Painting with an Expressive Virtual Chinese Brush, *IEEE Computer Graphics and Applications 24*, 5, 76-85.

CHU, N. S. H. AND TAI, C. L., 2005. MoXi: real-time ink dispersion in absorbent paper. In *ACM SIGGRAPH 2005 Papers,* Los Angeles, California, July 31 - August 04, 2005.

CURTIS, C., ANDERSON, S., SEIMS, J., FLEISCHER, K., AND SALESIN, D., 1997. Computer-Generated Watercolor, In *Proceedings of ACM SIGGRAPH 97*, ACM Press, 421-430.

DARDIS, O., AND MCCLOSKEY, J., 1998. Lattice Boltzmann scheme with real numbered solid density for the simulation of flow in porous media. *Phys. Rev. E: Lett. 57* (14), 4834–4837.

DAVIS, S. H, AND HOCKING, L. M., 2000. Spreading and imbibition of viscous liquid on a porous base. II. *Physics of Fluids 12*, 7, 1646-1655.

DESBRUN, M., SCHRODER, P., AND BARR, A., 1999. Interactive animation of structured deformable objects. Proc. of Graphics Interface'99, 1999.

EVERITT, C., 2001. Interactive Order-Independent Transparency, Nvidia white paper.

FERZIGER, J.H., AND PERIC , M., 1999. *Computational methods for fluid dynamics.* Springer-Verlag.

GELLER, S., M. KRAFCZYK, TOLKE, J., TUREK, S., AND HRON, J., 2006. Benchmark computations based on lattice-Boltzmann, finite element and finite volume methods for laminar flows." *Computers & Fluids* 35(8-9): 888-897.

GINZBURG, I., AND STEINER K. 2003. Lattice Boltzmann model for free surface flow and its application to filling process in casting, *J. Comput. Phys. 185*, 61–99.

GIRSHICK, R. 2004. Simulating Chinese Brush Painting: The Parametric Hairy Brush. Senior Honors Thesis, Department of Computer Science, Brandeis University, 2004.

GOLDSTEIN H. 1980. Classical Mechanics, 2nd ed., Addison-Wesley.

GUO, Q., AND KUNII, T. L., 1991. Modeling the diffuse painting of sumie, *IFIP Modeling in Computer Graphics,* 329-338.

GUO, Q., AND KUNII, T. L., 2003. Nijimi rendering algorithm for creating quality black ink paintings, in *Proceedings of Computer Graphics International 2003,* 152-159.

HAASE, C. S. AND MEYER, G. W.. 1992. Modeling pigmented materials for realistic image synthesis. ACM Trans. on Graphics, 11(4):305.

HAN, J. Y. 2005. Low-Cost Multi-Touch Sensing through Frustrated Total Internal Reflection. In Proceedings of the 18th Annual ACM Symposium on User Interface Software and Technology.

HANDPRINT FLATBRUSH 2007. Webpage retrieved July 2007, from http://www.handprint.com/HP/WCL/brush1.html

HANDPRINT WATERCOLOR 2007. Webpage retrieved July 2007, from http://www.handprint.com/HP/WCL/tech16.html

HARRIS, M., 2003. *Real-Time Cloud Simulation and Rendering*. Technical Report

#TR03-040, University of North Carolina.

HARRIS, M., 2004A. Course note 'Physically-Based Simulation on the GPU', SIGGRAPH 2004 Course 'GPGPU' (available at http://www.gpgpu.org/s2004/).

HARRIS, M., 2004B. Fast Fluid Dynamics Simulation on the GPU. In *GPU Gems*, R. Fernando, Ed., ch. 38, pp. 637–665. Addison Wesley.

HE, H.-S. (何懷碩), 2000. Brush-and-ink and the Abstraction of Chinese Painting. '筆墨與中國繪畫的抽象性'. In proceedings of 筆墨論辨：現代中國繪畫國際研討會. May 2000, Hong Kong. Also in *Art Cloud*. No. 54. P.7-16.

HE, X, AND LUO, L.-S., 1997. Lattice Boltzmann model for the incompressible Navier-Stokes equation, *J. Stat. Phys. 88*, 927-944.

HE, X., DOOLEN, G. D., AND CLARK, T., 2002. Comparison of the Lattice Boltzmann Method and the Artificial Compressibility Method for Navier-Stokes Equations, J. of Comput. Phys. 179, 439-451.

HERTZMANN, A. , JACOBS, C. , OLIVER, N. , CURLESS, B., AND SALESIN, D., 2001. Image Analogies. In *Proceedings of SIGGRAPH 2001 Conference,* ACM Press, 327-340.

HSU AND I.H.H. LEE 1994. Drawing and animation using skeletal strokes, SIGGRAPH '94 Proceedings, ACM Press, 1994, pp. 109 - 118.

IP, H. H. S., AND WONG, H. T. F., 1997. Calligraphic Character Synthesis using Brush Model. In CGI'97, Computer Graphics International conference, pp. 13-21, Hasselt-Diepenbeek, Belgium, June 23-27 1997.

JIN, X. CHEN, S., AND MAO, X., 2007. Computer-Generated Marbling Textures: A GPU-Based Design System. IEEE Computer Graphics and Applications. March/April 2007 (Vol. 27, No. 2), pp. 78-84.

JUDD, D. B., AND WYSZECKI, G., 1975 Color in Business, Science, and Industry. John Wiley & Sons, New York.

KANE C, REPETTO EA, ORTIZ M, MARSDEN JE., 1999. Finite element analysis of nonsmooth contact. Computer Methods in Applied Mechanics and Engineering 1999; 180:1 – 26.

KUBELKA, P. AND MUNK, F., 1931. Ein beitrag zur optik der farbanstriche. Z. tech Physik, 12:593.

KUNII, T. L., NOSOVSKIJ, G.V. AND HAYASHI, T., 1995. A diffusion model for computer animation of diffuse ink painting, Computer Animation, IEEE CS Press, 1995, pp. 98-102.

KUNII, T. L., NOSOVSKIJ, G. V., AND VECHERININ, V. L., 2001. Two-dimensional diffusion model for diffuse ink painting. *Int. J. of Shape Modeling, 7*, 1, 45-58.

LAERHOVEN, T., LIESENBORGS, J., AND REETH, F., 2004. Real-Time Watercolor Painting on a Distributed Paper Model. In *Proceedings of Computer Graphics International 2004*, 640-643.

LAERHOVEN, T. AND REETH, F., 2005. Real-time Simulation of Watery Paint, In *Journal of Computer Animation and Virtual Worlds*, 16:3-4 (Special Issue CASA2005), pages 429-439, Hong-Kong, China.

LAERHOVEN, T. AND REETH, F., 2007. Brush Up Your Painting Skills: Realistic Brush Design for Interactive Painting Applications. Accepted for publication in Visual Computer.

LALLEMAND, P. AND LUO, L., 2003. Theory of the lattice Boltzmann method: Acoustic and thermal properties in two and three dimensions. *Physical Review E*, 68:036706, 2003.

LEE, J., 1999. Simulating oriental black-ink painting, IEEE Computer Graphics and Applications, vol. 19, no. 3, IEEE CS Press, 1999, pp. 74–81.

LEE, J., 2001. Diffusion Rendering of Black Ink Paintings Using New Paper and Ink Models, *Computers and Graphics 25,* 2, 295-308.

LEFEBVRE, S., AND HOPPE, H., 2005. Parallel controllable texture synthesis. *SIGGRAPH 2005*, 777-786.

LEFEBVRE, S., AND HOPPE, H., 2006. Appearance-space texture synthesis. *SIGGRAPH 2006*, 541-548.

LEI, S. I. E., AND CHANG, C.-F., 2004. Real-Time Rendering of Watercolor Effects for Virtual Environments. *Advances in Multimedia Information Processing - PCM 2004*, Lecture Notes in Computer Science, Volume 3333/2004, Springer Berlin/Heidelberg.

LUM, E., AND MA, K.L., 2001. Nonphotorealistic Rendering using Watercolor Inspired Textures and Illumination. Pacific Graphics 2001, October 16-18, 2001.

MAO, X., SUZUKI, T., AND IMAMIYA, A., 2003. AtelierM: A Physically Based Interactive System for Creating Traditional Marbling Textures. In *Proc. 1st Int'l Conf. Computer Graphics and Interactive Techniques in Australasia and South East Asia*, ACM Press, 2003, pp. 79-86.

MARK, W. R., GLANVILLE, R. S., AKELEY K., AND KILGARD, M. J., 2003. Cg: a system for programming graphics hardware in a C-like language. *ACM Transactions on Graphics,* 22, 3, 896-907.

MATHWORLD SUPERELLIPSE 2007. Webpage retrieved July 2007, from http://mathworld.wolfram.com/Superellipse.html

MAURER-MATHISON, D., 1999. The Ultimate Marbling Handbook: A Guide to Basic and Advanced Techniques for Marbling Paper and Fabric, Watson-Guptill.

MERIAM, J. L. AND KRAIGE, L. G., 1992. Engineering Mechanics: Statics. John Wiley & Sons, Inc., 3rd edition.

MI X., XU J., TANG M., AND DONG J., 2002. The droplet virtual brush for Chinese calligraphic character modeling. in: Proceedings of Applications of Computer Vision, 2002.

MILENKOVIC, V. J. AND SCHMIDL, H. 2001. Optimization-based animation. In *Proceedings of the 28th Annual Conference on Computer Graphics and interactive Techniques* SIGGRAPH '01. ACM Press, New York, NY, 37-46.

NG, H. N. AND GRIMSDALE, R. L., 1996. Computer graphics techniques for modeling cloth. *IEEE Computer Graphics and Applications*, 16(5):28-41, September 1996.

ONESTROKE 2007. Webpage retrieved July 2007, from http://onestroke.com

PAINTER 2007. Software package by Corel Corporation (http://www.corel.com).

PANDOLFI A., KANE C., MARSDEN, J. E. AND ORTIZ, M., 2002. Time-discretized variational formulation of non-smooth frictional contact, International Journal for Numerical Methods in Engineering, vol. 53, 2002, pp. 1801-1829.

PESHKIN, M. A., AND SANDERSON, A. C.,1988. Minimization of Energy in Quasistatic Manipulation, *IEEE Transactions on Robotics and Automation*, 4:5, 1988.

PHAM, B. 1991. Expression brush strokes, CVGIP: Graphical Models and Image Processing, vol. 53, No. 1.

PLANTE, E., CANI, M.-P., POULIN, P., 2001. A Layered Wisp Model for Simulating Interactions Inside Long Hair in Computer Animation and Simulation '01.

QIU, F., ZHAO, Y., FAN, Z., WEI, X., LORENZ, H., WANG, J., YOAKUM-STOVER, S., KAUFMAN, A., AND MUELLER, K. 2004. Dispersion Simulation and Visualization For Urban Security. In *Proceedings of the Conference on Visualization '04* (October 10 - 15, 2004).

RADOVITZKY R, ORITIZ M., 1999. Error estimation and adaptive meshing in strongly nonlinear dynamic problems. Comput Methods Appl Mech Engrg, 1999, 172:203～240.

RASKAR, R., 2001. Hardware support for non-photorealistic rendering. Graphics Hardware, LA.

RUDOLF, D., MOULD, D., NEUFELD, E., 2003. Simulating Wax Crayons. *Proceedings of the 11th Pacific Conference on Computer Graphics and Applications*, pp. 163–172.

SAITO S. AND NAKAJIMA, M. 1999. 3D physics-based brush model for painting, SIGGRAPH 99 Sketches, Conference Abstracts and Applications, ACM Press, 1999, p. 266.

SILBERGELD, J., 1982. *Chinese Painting Style: Media, Methods, and Principles of Form.* University of Washington Press.

SMITH, A. R., 2001. Digital paint systems: An anecdotal and historical overview. *IEEE Annals of the History of Computing*, 23(2).

SONG, O., SHIN, H., AND KO, H. 2005. Stable but nondissipative water. *ACM Trans. Graph.* 24, 1 (Jan. 2005), 81-97.

STAM, J. 1999. Stable Fluids, In *Proceedings of ACM SIGGRAPH 99*, ACM Press, 121-128.

STRASSMANN, S. 1986. Hairy Brushes, SIGGRAPH 1986 Proceedings, ACM Press, 1986, pp. 225 - 232.

STRENGERT, M., KLEIN, T., BOTCHEN, R. P., STEGMAIER, S., CHEN, M., AND ERTL, T., 2006. Spectral Volume Rendering using GPU-based Raycasting. *Visual Computer*, 22(8):550-561.

SUCCI, S., 2001. *The lattice Boltzmann equation for fluid dynamics and beyond.* Oxford University Press.

SUN, N.-Z., 1996. *Mathematical modeling of groundwater pollution.* Springer-Verlag.

SWIDER, J. R, HACKLEY, V. A., AND WINTER, J., 2003. Characterization of Chinese Ink in size and surface, *J. of Cultural Heritage 4*, 175-186.

TABARROK, B. AND RIMROTT, F.P.J, 1994. Variational Methods and Complementary Formulations in Dynamics, Kluwer Academic.

THUEREY, N., 2003. *A single-phase free-surface lattice-Boltzmann method.* Master-thesis, Erlangen Germany.

WEI, L.-Y., 2004. Tile-Based Texture Mapping on Graphics Hardware, *SIGGRAPH/EUROGRAPHICS Conference on Graphics Hardware 2004*.

WEI, X., ZHAO, Y., FAN, Z., LI, W., QIU, F., YOAKUM-STOVER, S., AND KAUFMAN, A.,

2004. Lattice-Based Flow Field Modeling. *IEEE Transactions on Visualization and Computer Graphics*, 10, 6, 719-729.

WEISKOPF, D., 2004. "Dye Advection without the Blur: A Level-Set Approach for Texture," Computer Graphics Forum, Proc. Eurographics 2004, vol. 23, no. 3, pp. 479-488, 2004.

WHITAKER, R. T., 2002. *Isosurfaces and Level-Set Surface Models.* Technical report, School of Computing, University of Utah.

WIKIPEDIA 3DPRINT 2007. Webpage retrieved July 2007, from http://en.wikipedia.org/wiki/3D_printing

WIKIPEDIA ALEXANDER 2007. Webpage retrieved July 2007, from http://en.wikipedia.org/wiki/William_Alexander_(television_painter)

WIKIPEDIA ANTI-ALIASING 2007. Webpage retrieved July 2007, from http://en.wikipedia.org/wiki/Anti-aliasing

WIKIPEDIA GICLÉE 2007. Webpage retrieved July 2007, from http://en.wikipedia.org/wiki/Gicl%C3%A9e

WIKIPEDIA MARBLING 2007. Webpage retrieved July 2007, from http://en.wikipedia.org/wiki/Paper_marbling

WIKIPEDIA NUMERICAL DIFFUSION 2007. Webpage retrieved July 2007, from http://en.wikipedia.org/wiki/Numerical_diffusion

WIKIPEDIA RASTER 2007. Webpage retrieved July 2007, from http://en.wikipedia.org/wiki/Raster_graphics

WIKIPEDIA ROSS 2007. Webpage retrieved July 2007, from http://en.wikipedia.org/wiki/Bob_Ross

WONG, H.T.F. AND IP, H.H.S. 2000. Virtual brush: a model-based synthesis of Chinese calligraphy, Computers and Graphics, vol. 24, no. 1, 2000, pp. 99-113.

WYNN, C., 2002. OpenGL Render-to-texture. Game Developers' Conference 2002.

XU, S., LAU, F.C.M., TANG, F., AND PAN, Y.H., 2003. Advanced Design for a Realistic Virtual Brush, Computer Graphics Forum, vol. 22, no. 3, 2003, pp. 533-542.

XU, S., TANG, M., LAU, F.C.M., AND PAN, Y.H., 2002. A solid model based virtual hairy brush, Computer Graphics Forum, vol. 21, no. 3, 2002, pp. 299-308.

YOSHIDA, S., KURUMISAWA, J., NOMA, H., TETSUTANI, N. AND HOSAKA, K., 2004. Sumi-Nagashi: Creation of New Style Media Art with Haptic Digital Colors, ACM Multimedia 2004, pp. 636-643.

YU, D., MEI, R., LUO, L.-S., AND SHYY, W., 2003. Viscous flow computations with the method of lattice Boltzmann equation, *Progress in Aerospace Science 39*, 329-367.

ZHANG, Y. SATO, J. TAKAHASHI, K. MURAOKA AND N. CHIBA. 1999. Simple cellular automation-based simulation of ink behavior and its application to Suibokuga-like 3D rendering of trees, Journal of Visualization and Computer Animation, 1999.