A FRAMEWORK FOR DIGITAL WATERCOLOR

A Thesis

by

PATRICK O'BRIEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

August 2008

Major Subject: Visualization Sciences

A FRAMEWORK FOR DIGITAL WATERCOLOR

A Thesis

by

PATRICK O'BRIEN

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

Approved by:

Chair of Committee,	Donald House
Committee Members,	Richard Davison
	John Keyser
Head of Department,	Tim McLaughlin

August 2008

Major Subject: Visualization Sciences

ABSTRACT

A Framework for Digital Watercolor.

(August 2008)

Patrick O'Brien, B.B.A., University of St. Thomas

Chair of Advisory Committee: Dr. Donald House

This research develops an extendible framework for reproducing watercolor in a digital environment, with a focus on interactivity using the GPU. The framework uses the lattice Boltzmann method, a relatively new approach to fluid dynamics, and the Kubelka-Munk reflectance model to capture the optical properties of watercolor. The work is demonstrated through several paintings produced using the system.

DEDICATION

I dedicate this thesis to my family and all those who have helped me throughout the years.

ACKNOWLEDGEMENTS

I would like to thank my committee chair, Dr. House, and my committee members, Dr. Keyser, and Prof. Davison, for their guidance and support throughout the course of this research. I would especially like to thank Dr. House for his guidance both before and during my time at A&M. He is an exceptional teacher, and a great person.

Thanks also go to my friends and colleagues and the department faculty and staff for making my time at Texas A&M University a great experience. I extend a great amount of gratitude to the faculty, staff, and students in the Visualization Sciences program for making my time there an incredible experience. I would like to thank Michael Losure, Seth Freeman, and Tony Piedra for their help and collaboration throughout my time in the Visualization Sciences program. I also give thanks to Mayank Singh, my GPU buddy.

Finally, thanks to my mother, father, brother, and wife for their encouragement, patience and love.

TABLE OF CONTENTS

LIST OF FIGURES

FIGURE Page

CHAPTER I

INTRODUCTION

Simulating paint digitally is an exciting area of research. Digital painting brings together the advantages of computers with the beauty of traditional media. Computers offer a key advantage over traditional media. They can be much more forgiving when users make mistakes. Most computer programs allow a user to undo previous actions. Removing an errant paint stroke is as easy as pushing a button. Painters can also save their work at any time while painting. Another key feature of digital painting is the ability to manipulate the painting using controls. For example, adjusting the rate at which paint dries. Digital painters use controls to achieve effects not possible in traditional media. Digital painting's properties offer artists more freedom to experiment in their work.

Watercolor is a popular painting technique known for several unique characteristics. The combination of water and pigments, applied to paper, creates interesting patterns and shapes common only to watercolor. The medium is also quite distinctive due to its vibrant colors and transparent luminous quality. Perhaps the most distinctive quality of watercolor is its spontaneity. Putting brush to paper often creates unpredictable results as water, pigments and paper interact. This property of watercolor makes the medium fascinating, but also difficult for beginners.

This thesis follows the style of *IEEE Transactions on Visualization and Computer Graphics*.

Unlike other media, watercolor is not very forgiving to mistakes. Watercolor's unique properties make creating a digital watercolor tool a challenge.

There have been many attempts at creating digital watercolor tools using both image-based and physically-based methods. Image-based methods use image operations and textures to re-create the look of watercolor. Physically-based modeling accounts for physical dynamics and is a way to create realistic appearing digital models of physical phenomena. The non-photorealistic community generally agrees physically-based approaches provide the best results [1, 2]. Research has demonstrated fluid movement, brush and paper interaction, and light and surface interaction in the computer with fairly convincing results. For example, Curtis et al. [3] developed an offline physically-based watercolor tool, and more recently Chu and Tai [10] introduced a physically-based eastern ink tool that offers several new advancements in fluid modeling. However the interaction of water, pigments, and paper is complex and slow to compute. Thus, there is still work to be done in capturing the complex nature of watercolor within an interactive painting tool.

The goal of this thesis is to develop and demonstrate a real-time watercolor tool that is easily extendible. The Graphics Processing Unit (GPU) holds promise of making real-time watercolor a possibility. Many new research developments in watercolor have used the GPU, however all of these tools use the mathematically complex Navier-Stokes equations [5] for simulation of fluid flow. A different set of equations, called the lattice Boltzmann method (LBM) [5], offers a simpler mathematical model of that flow. The LBM is a cellular based model, making it ideal for GPU implementation, which itself

has a spatially distributed parallel processing architecture. In addition to mapping well to the GPU, the LBM also makes it easy to add new physics that are hard to describe macroscopically, and therefore directly supports the goal for an extensible interactive tool.

CHAPTER II

BACKGROUND

Reproducing watercolor digitally requires knowledge of how traditional watercolor effects form and prior research on computer-generated watercolor.

A. Characteristics of Watercolor

Physically, the behavior of watercolor includes the interaction of pigments flowing in water, the absorption of pigments and water into paper, and the evaporation of water from the paper [3]. Watercolor paper is typically made from cotton or linen rags to avoid buckling. It can be described as being mostly air laced with a web of tangled rag fibers, that creates a highly absorbent material. Usually watercolor paper is treated with sizing to slow water absorption and diffusion. Watercolor pigment is made of finely ground particles. The particles are mixed with gum for body and glycerin for viscosity. The glycerin also binds colorant to the paper. Pigments have four important properties that determine their behavior: density, staining power, granulation, and flocculation. Density determines how long a pigment stays suspended in the water and consequently how far it will spread. Staining power is an estimate of a pigment's tendency to adhere to the paper. Granulation describes whether a pigment settles into spaces in rough paper. Finally, flocculation accounts for electrical effects drawing pigments together into clumps.

The size and bristle structure of the watercolor brush play an important role in watercolor. Watercolor brushes tend to be softer and hold more water than brushes used for other painting methods. The size of a brush and its bristles determine its footprint. A brush footprint is the contact area between the brush and paper. The footprint determines how much water and pigment are deposited onto the paper. Typical brush techniques include dry-brush, wet-on-dry and wet-in-wet. Wet-in-wet is a technique where a paintbrush loaded with water and pigment is applied to paper saturated with water so that the paint can spread freely on the paper [2]. Dry-brush involves applying a brush with paint and a small amount of water to dry paper [2]. Wet-on-dry is a typical painting technique using a wet brush loaded with paint on dry paper. Fig. 1 shows the effects that can be produced using these techniques. Dry-brush (Fig. 1a) will only leave paint on raised portions of rough paper. Wet-on-dry creates an effect known as edge darkening. Edge darkening (Fig. 1b) happens as water at the edge of a brushstroke dries faster than the inside. Water at the inside migrates towards the outside carrying pigments to create more pigment deposition at the outside of the brushstroke. Wet-in-wet creates effects such as back-runs (Fig. 1c), granulation (Fig. 1d), and flow patterns (Fig. 1e). Back-runs occur when a puddle of water spreads back into a damp region. Granulation occurs as pigments settle into the hollows of the paper. While this effect is not strictly related to wet-in-wet, it is strongest when the paper is very wet. Flow patterns are a result of brushstrokes spreading freely on the paper. The effect creates soft feathery shapes that follow the direction of water flow. A final technique is glazing (Fig. 1f) which is the process of painting thin washes of paint one over the other after each one dries.

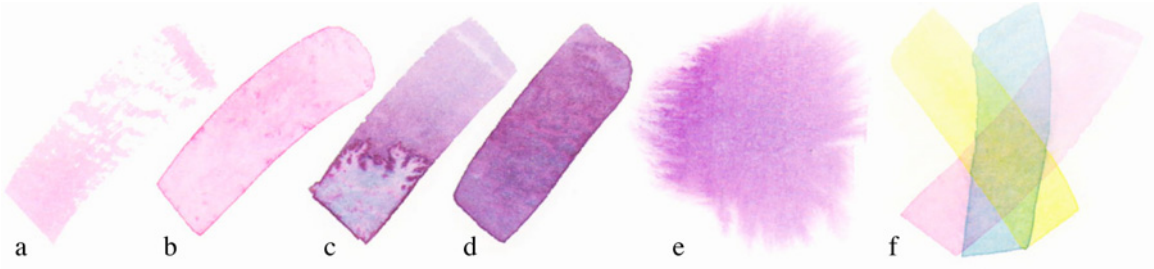

Fig. 1: Watercolor Effects from Curtis et al. [3] (a) dry-brush, (b) edge darkening, (c) back-runs, (d) granulation, (e) flow patterns (f) glazing.

The result is a luminous appearance as the layers mix optically.

B. Computer Generated Watercolor

There have been several contributions in reproducing watercolor on the computer. Small was probably the first to suggest using a cellular automata method for simulating watercolors [7]. The simulation attempted to predict the behavior of pigment and water when applied to paper. While not a real-time tool, it served as a basis for future watercolor tools. Building on Small's work, Curtis et al. [3] suggested a physically based model capable of producing several real watercolor effects including dry brush, edge darkening, back runs, granulation, flow effects and glazing. The model uses three layers. A shallow water layer is used to move the water and pigment across the paper. Pigment is then deposited in the pigment-deposition layer. The final layer represents water absorbed into the paper and diffused by capillary action. The simulation solves a form of the shallow water equations for fluid flow. Curtis uses the Marker-And-Cell (MAC) [8] method to solve the shallow water equations [5] based on Foster and Metaxas [9] work.

The final painting consists of washes or glazes composited using the Kubelka-Munk [10] diffuse reflectance model. The model proposed by Curtis is mostly suited for automatic rendering, as opposed to interactive rendering, due to computational complexities.

Laerhoven and Reeth [11] further the work done by Curtis by making a more interactive watercolor system. They suggest a physically based model similar to the one proposed by Curtis. The main difference in their method is the use of the Graphics Processor Unit (GPU) and the way the fluid is computed. Laerhoven and Reeth use Stam's [12] approach to fluid-flow on the GPU. Stam's method uses an implicit backwards-Euler integration, making the simulation more stable at higher viscosity and allowing larger time steps to speed up the simulation. Like Curtis, they use the Kubelka-Munk reflectance model for rendering, however they solve the model's equations on the GPU.

Burgess et al. [13] suggest a different non-physically based approach to watercolor rendering. The system takes 3D models and makes them look like they were painted with watercolor. Burgess et al. use three layers of paint to achieve a watercolor look: a diffuse layer which is the pigment color in uniform thickness, a shadow layer, and a texture layer where pigments have varying thickness. Post-processing is used to create edge darkening and imperfect object shape.

More recently, Chu and Tai [10] present a new physically based method for simulating Eastern Ink. Eastern Ink shares many qualities similar to watercolor as demonstrated in Fig. 2. Flow patterns (Fig. 1e and Fig. 2a) and edge darkening

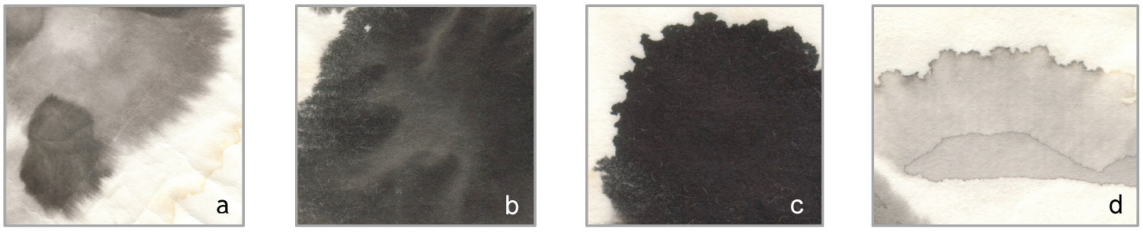

Fig. 2: Ink Effects from Chu and Tai [4] (a) dry-brush, (b) edge darkening, (c) back-runs, (d) granulation, (e) flow patterns (f) glazing.

(Fig. 1b and Fig. 2d) are effects common to both eastern ink and watercolor. Additionally, branching patterns (Fig. 2b) combined with boundary roughening (Fig. 2c) are similar to back-runs (Fig. 1c). Their simulation uses the lattice-Boltzmann method [5] for solving fluid flow. Chu and Tai's work provides several contributions to previous work including parallel GPU processing, shape evolution of fluid flow, and medium permeability. Like Curtis, Chu and Tai use a three-layer paper model: a surface layer for pigment deposition onto the paper, a flow layer for pigment and water flow on the paper and a fixture layer for pigment deposited in the paper as ink dries. The system makes use of the GPU for the fluid simulation. The lattice Boltzmann method is ideal for the GPU due to its use of simple local operations at each lattice site.

C. Lattice Boltzmann Method

The lattice Boltzmann method has its roots in the Boltzmann equation [14], proposed in 1872 by Ludwig Boltzmann. The Boltzmann equation describes the behavior of gas on a microscopic level using kinetic theory [5]. It gives a statistical distribution of particles in

a single-particle phase space. In 1976, Hardy, Pomeau, and Pazzis [15] proposed the Lattice Gas Cellular Automata Method (LGCA) as depicted in Fig. 3. The LGCA was introduced as a conceptual model for the microscopic behavior of fluid, capable of solving the Navier Stokes equation [5]. The model is composed of a lattice where each site is a boolean value indicating the particle state as shown in Fig. 3. In Fig. 3, a site occupied by particles has a value of 1 and a site with no particles has a value of 0. Two processes occur at a site, propagation and collision of particles. In propagation particles move in the direction of their velocity to the neighboring site. The collision step resolves sites that receive multiple particles after streaming. As Fig. 3 illustrates, the particles' velocity vectors are rotated 90 degrees to avoid the collision. A main issue

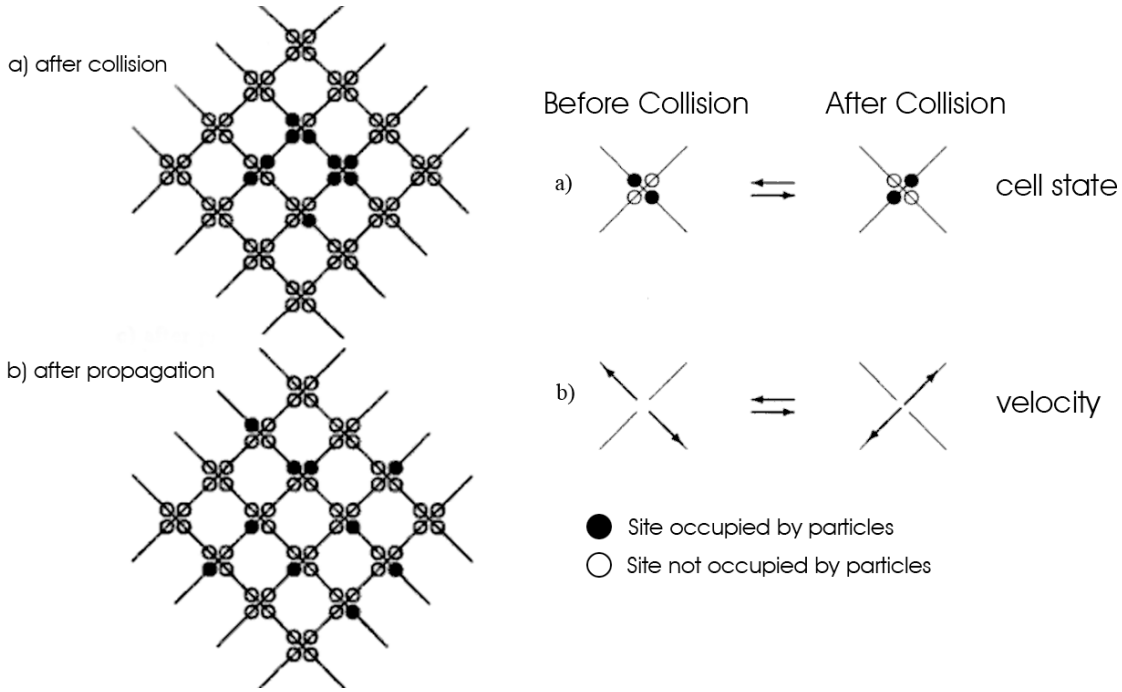

Fig. 3: Lattice Gas Cellular Automata

with the LGCA is it is highly anisotropic due to rotational invariance. This simply means vortices produced by the model are square shaped [5]. In 1986, Frisch et al. [16] introduced the hexagonal Lattice Gas Cellular Automata Method (LGCA), which solved some of the anisotropy issues. However despite Frisch's efforts, several problems still plagued the Lattice Gas Cellular Automata method. The problems included large fluctuations in the fluid flow (statistical noise), an inability to simulate in three dimensions, and simulations were limited to highly viscous fluids [5]. The lattice Boltzmann method (LBM) arose in response to the limitations of the LCGA. First proposed by McNamara and Zanetti [17], the lattice Boltzmann method replaced the boolean particle number in a lattice direction with the density distribution function to reduce statistical noise. Unfortunately, the LBM still suffered from problems when simulating 3D flow and could only simulate viscous fluids. The practical viability of simulating in three dimensions came with Higuera and Jimenez [18]. They suggest changes to the collision process turning the nonlinear collision operator into a linear operation. These changes made fluid simulations perform faster allowing 3D simulations. Higuera et al. [19] suggest enhanced collisions for the LBM that allow simulations with low viscous fluids. They eliminate collisions from the LBM so that only the consequence of collisions matters. Quian et al. [20] suggest a final improvement to the collision operator known as the Bhatnagar-Gross-Krook approximation. This version of the LBM is known as the lattice-BGK model (LBGK) and provides a single time relaxation. The LBGK is the most popular LBM used today due to its simplicity and efficiency [5]. The LBM is inherently compressible [5]. Consequently, it models the

compressible Navier-Stokes equation. Fluid compressibility is a main feature of the LBM and is what gives it a performance advantage over other methods [4]. However, He and Luo [21] recognized there is also a need for incompressible fluid and introduced an incompressible variant of the lattice Boltzmann model. One limitation that comes with the incompressible LBM is the fluid speed must be kept low in order to minimize the compressibility effect.

The lattice Boltzmann methods work on a lattice, and are a type of cellular automaton. Fig. 4 shows that in a cellular automaton model all cells are updated at each time step according to rules that take into account the surrounding cells. The interaction of the cells determines the complex behavior of the automaton. Several variations of the LBM exist, and are named DXQY, where X is the dimension and Y is the number of lattice velocities or vectors [22]. Fig. 5 depicts a cell from a D2Q9 lattice. A lattice vector is referred to as $\mathbf{e}_i$ where i is the lattice vector number. In Fig. 5 the lattice vectors are $\mathbf{e}_0 - \mathbf{e}_8$. At each lattice site **x** and time *t*, fluid particles moving at

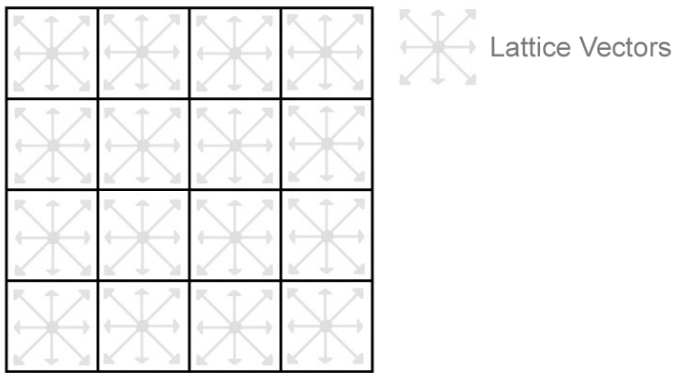

Fig. 4: A Cellular Automata : Lattice Boltzmann Method

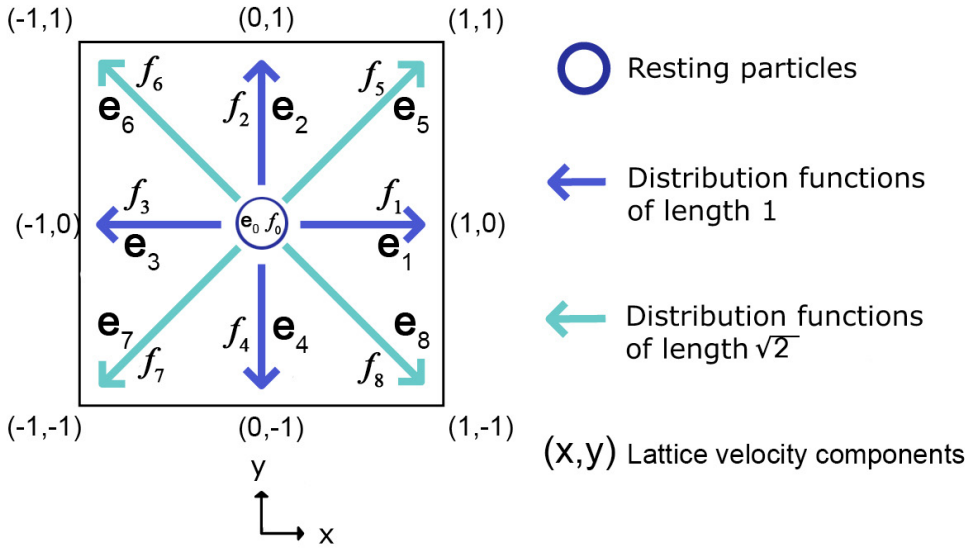

Fig. 5: D2Q9 Lattice Cell Based on Fig. from N. Thurey [22]

arbitrary velocities are modeled by particle distribution functions $f_i(\mathbf{x},t)$ [4]. Each $f_i(\mathbf{x},t)$ is the expected number of particles moving along a lattice vector $\mathbf{e}_i$. Each side of the cell has lattice unit equal to 1. The magnitude of velocity vectors $\mathbf{e}_1$ through $\mathbf{e}_4$ is 1 lattice unit per time step. The magnitude of velocity vectors $\mathbf{e}_5$ through $\mathbf{e}_8$ is $\sqrt{2}$ lattice units per time step [23]. The magnitude of vector $\mathbf{e}_0$ is 0, because it represents particles at rest. Resting particles do not move in the next time step, but may be accelerated due to collisions. As a result, the number of resting particles can change.

The cell density and overall speed and direction that the particles in the cell move are calculated using a cell's particle distribution functions. The density

$$\rho = \sum_{i=0}^{8} f_i \qquad (1)$$

is the sum of all particle distribution functions. The velocity

$$\mathrm{u} = \frac{1}{\rho_0} \sum_{i=1}^{8} \mathrm{e}_i f_i \qquad (2)$$

is the sum of the lattice vectors $\mathbf{e}_i$ times the corresponding distribution function $f_i(\mathbf{x},t)$. The initial density ρ_0 is usually set to 1.

A simulation consists of two steps: streaming and collision. These two steps simulate the convection and diffusion phenomena that occur on a macroscopic level in physics. During the streaming step, the particles move from one cell to the next. For instance, $\text{cell}_{i,j}$'s distribution function for the lattice vector pointing downwards would be copied to $\text{cell}_{i+1,j}$'s distribution function for the lattice vector pointing downwards. The lattice vector in the center does not point anywhere and so the "at rest" particles are not copied. Fig. 6 graphically displays the streaming step. The streaming step is described mathematically as

$$f_i\left(x + e_i \Delta t, t + \Delta t\right), \qquad (3)$$

where $\mathbf{e}_i$ is the lattice vector pointing in the opposite direction of the distribution

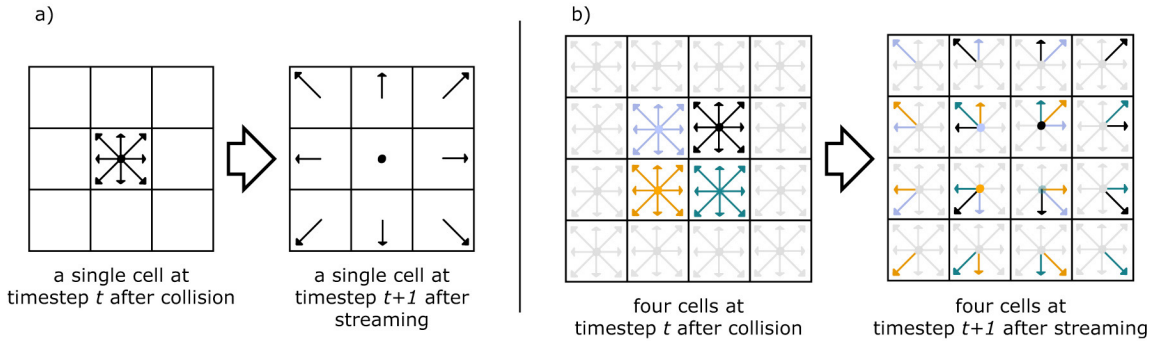

Fig. 6: Streaming Step from N. Thurey [22]

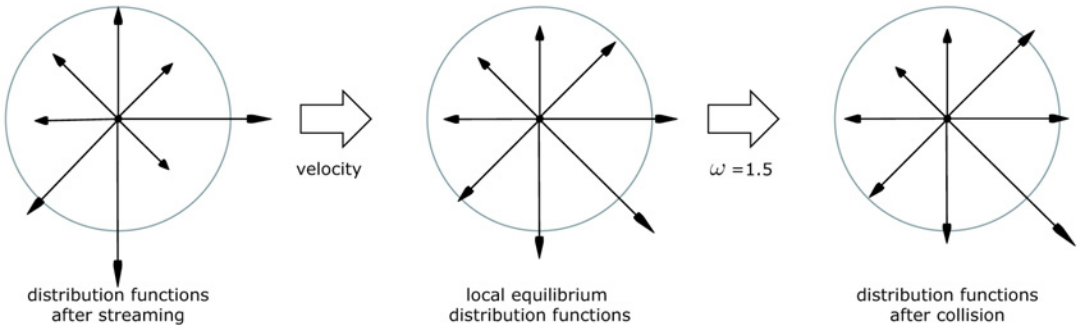

Fig. 7: Collision Step from N. Thurey [22]

function. For example, if the distribution function is f_1, then the lattice vector would be $\mathbf{e}_3$ or (-1,0). The lattice Boltzmann method requires interfacial boundary conditions to determine the distribution functions at boundary sites. A variety of boundary conditions exist for determining the distribution functions including periodic, Von Neumann, Dirichlet, and bounceback [5]. The most common boundary conditions for the lattice Boltzmann method are "no-slip" walls such that fluid close to the boundary does not move [22]. This amounts to each cell next to a boundary having the same amount of particles moving into the boundary as moving in the opposite direction.

In the collision step, particles arrive at a lattice site and collide with other particles. Fig. 7 depicts this step graphically. As Fig. 7 demonstrates, the collision step does not change the density or velocity of a cell. It only changes the distribution of particles for all particle distribution functions in a cell [22]. For instance, consider a $\text{cell}_{i,j}$ where the fluid moves along the positive x-axis. The cell will not lose any particles during collision. However the movement will be scattered to other cells' lattice vectors that point in the direction of the positive x-axis. Lattice vectors pointing in the opposite

direction will become smaller. This is illustrated in Fig. 7. In the next stream step, neighboring cells with an x_{i+1} coordinate will receive a slightly larger particle distribution function from $cell_{i,j}$, while neighboring cells with an x_{i-1} coordinate will receive smaller distribution functions. To model this behavior, the equilibrium distribution function, $f_i^{(eq)}$ and new distribution functions must be calculated. He and Luo [21] suggest that the equilibrium distribution function

$$f_i^{(eq)} = w_i\left(\rho + \rho_0\left(\frac{3}{c^2}\mathrm{e_i}\cdot\mathrm{u} + \frac{9}{2c^4}(\mathrm{e_i}\cdot\mathrm{u})^2 - \frac{3}{2c^2}\mathrm{u}\cdot\mathrm{u}\right)\right) \qquad (4)$$

works well in reproducing incompressible flow behavior. Each lattice vector has an equilibrium distribution function. The weights w_i can be interpreted as different masses of the particles moving along the different lattice directions [5]. The weights for a D2Q9 lattice are 4/9 for i=0, 1/9 for i=1,2,3,4 and 1/36 for i=5,6,7,8. The basic speed on the lattice is denoted by c [23]. In basic implementations $c = \rho_0 = 1$. The LBM has built in advection and the term $\left(\frac{3}{c^2}\mathrm{e_i}\cdot\mathrm{u} + \frac{9}{2c^4}(\mathrm{e_i}\cdot\mathrm{u})^2 - \frac{3}{2c^2}\mathrm{u}\cdot\mathrm{u}\right)$ in the equilibrium distribution function is responsible for the advection. The new distribution functions are

$$f_i' = (1-\omega)f_i + \omega f_i^{(eq)}. \qquad (5)$$

The relaxation rate ω determines the viscosity of the fluid and affects how quickly the fluid reaches equilibrium. For $\omega < 1$ the fluid will be more viscous like honey while $\omega > 1$ will produce less viscous fluids like water.

In the literature, the streaming and collision steps are often combined into one formula known as the lattice Boltzmann equation,

$$f_i(\mathrm{x} + \mathrm{e_i}\Delta t, t + \Delta t) = (1-\omega)f_i(\mathrm{x}, t) + \omega f_i^{(eq)}(\mathrm{x}, t) \qquad (6)$$

The left side of the lattice Boltzmann equation describes the streaming step, while the right side describes the current distribution function and local equilibrium [22].

D. Kubelka-Munk Reflectance Model

The Kubelka-Munk Reflectance model is a physically based model that simulates the scattering and absorption of light by materials. The model assumes that light scatters at a single point, and the resulting subsurface scattering is either diffuse or shaped by the scattering properties of the material [25]. The KM model uses an absorption coefficient K and scattering coefficient S to model light scattering. Theses coefficients can be derived experimentally using spectral measurements or set interactively in an application. Interactively deriving the two coefficients is more convenient, because there is no need for equipment measuring spectral properties. Curtis suggests using the following equations for K and S:

$$S = \frac{1}{b}\coth^{-1}\left(\frac{b^2 - (a - R_w)(a-1)}{b(1-R_w)}\right) \tag{7}$$

$$K = S(a-1) \tag{8}$$

where

$$a = \frac{1}{2}\left(R_w + \frac{R_b - R_w + 1}{R_b}\right), \quad b = \sqrt{a^2 - 1}\ . \tag{9}$$

R_w represents a “unit thickness” of a pigment over white and R_b represents a “unit thickness” of a pigment over black. Both R_w and R_b are specified as RGB triples. Curtis

requires $0 < R_b < R_w < 1$. The computations of K and S are performed on each of the R, G and B color channels independently.

Once K and S are found, a layer's reflectance R and transmittance T are given by

$$\begin{aligned} R &= \sinh bSx / c \\ T &= b / c \end{aligned} \qquad \text{where} \quad c = a\sinh bSx + b\cosh bSx \; . \tag{10}$$

The thickness of a pigment layer is denoted by x. Given two layers with reflectance R_1 and R_2 and transmittances T_1 and T_2, the overall reflectance and transmittance is

$$R = R_1 + \frac{T_1^2 R_2}{1 - R_1 R_2} \qquad T = \frac{T_1 T_2}{1 - R_1 R_2} \; . \tag{11}$$

Note that in general $R_1 R_2 \neq R_2 R_1$.

While Kubelka-Munk theory has been discussed in computer graphics, Haase and Meyer [23] are the first to use the theory to solve color synthesis problems. Their work derives the equations for modeling Kubelka-Munk theory in computer graphics and shows why the Kubelka-Munk reflectance model is necessary for capturing the optical effects that occur when mixing pigments. Haase and Meyer prove additive (RGB) and subtractive (CMY) color spaces are inadequate for modeling pigmented materials, as shown in Fig. 8. This is because pigmented materials are opaque particles in a transparent medium. Fig. 8 demonstrates the importance of wavelength samples in the accuracy of the Kubelka-Munk model. However, even the 3 sample model works better than the RGB color space.

Recently Donner and Jensen [25] made improvements to Kubleka-Munk theory by making a variant of the Kubelka-Munk model in frequency space. This variant produces more realistic results in layered translucent materials. Donner and Jensen [25]

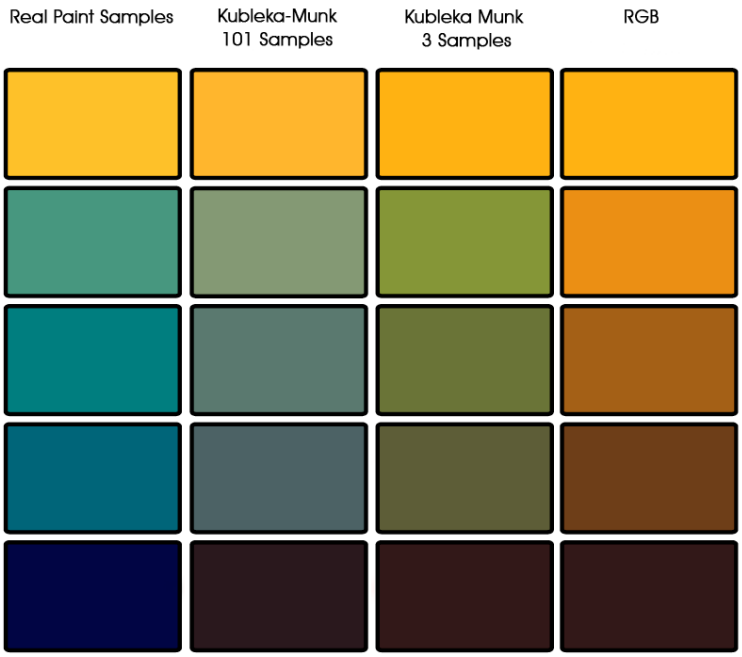

Fig. 8: Comparison of Kubelka-Munk to RGB based on Baxter et al. [29]

introduce a multipole diffusion approximation to scattering of light at a surface. Diffusion approximation is a way to solve the Bidirectional Scattering Surface Reflectance Distribution function (BSSRDF) [26] used in physically based calculations of subsurface scattering.

Many paint programs [3,27,13,27,11,29,30] use Kubelka-Munk theory to reproduce the optical effects of paint. Baxter et al. [30] were the first to introduce an interactive version of the Kubelka-Munk model by solving the equations on the GPU. Most current paint programs [27,11,29,30] now use the GPU to solve the Kubelka-Munk equations since it allows for interactive programs.

E. Fluid Simulation on the GPU

There are two types of GPU programs typically used in GPU processing [31], vertex and fragment programs. The vertex program involves operations occurring at the vertex such as lighting and transformations. The fragment program, involves operations like reading from texture memory and applying texture values at fragments, which is a per-pixel data structure created by the rasterization of graphics primitives [31]. Both types of programs are compiled and linked into executable code that runs on the GPU.

A typical GPU based approach to fluid dynamics involves integrating shaders written in some shading language with a high level programming language. A Graphics Application Programming Interface (API) provides the bridge between the high level programming language and the shading language. The API allows the program to pass data to and from the shaders.

Physically Based Simulations performed on the GPU are typically referred to as General Purpose Computation on GPU (GPGPU). Most fluid simulations on the GPU use a grid of cells. Ideally, each fragment should be a cell in the grid. This is accomplished using a screen size quad with a one to one mapping between pixels and texels. Current GPUs do not allow both reading and writing to the same texture, because the reading and writing mechanisms are independent of each other. Allowing reading and writing to the same buffer would require a highly synchronized approach to avoid overwriting values, which would reduce the efficiency of the GPU [32]. An approach called *Ping Pong* is used to circumvent this limitation. *Ping Pong* uses two textures to represent one set of data. During one iteration or pass of a shader, one texture is used as

the read texture and the other is used as the write texture. After the shader finishes execution, the textures are swapped, making the write texture the read texture the next time the shader runs. This process is repeated until the shader is disabled.

Harris et al. [33] discuss the many advantages of using the Graphics Processing Unit for simulations. They also discuss common operations a GPU can perform, such as computing directional forces, convection, and boiling. They point out that GPUs are well suited for simulations, due to their parallel nature, the speed of performing imaging operations compared to Central Processing Units (CPU), and the ability to balance the many processing tasks in a simulation between the CPU and GPU for interactive simulations. A GPU does have disadvantages, most notably low precision. Currently, GPUs use 8-bit integer precision, which is only one quarter of the precision offered in a CPU.

Wei et al. [34] implement the lattice Boltzmann equations on the GPU. They suggest placing all packet distributions with the same direction in one texture to avoid the overhead of switching between textures. Another trick is to pack four packet distributions from different directions into one RGBA texel which reduces the memory requirement of distributions by one-fourth. They overcome the precision limitations of GPUs by using range scaling. Range scaling avoids clamping errors and takes full advantage of the hardware precision by mapping all variables to between [-1, 1] [34].

CHAPTER III

METHODOLOGY

A modular design is used in the watercolor program so that the program can be easily improved upon in the future. The program is separated into models of how the brush, pigment, paper, and water behave. The individual models are designed so that a change to one will require little change in the others. The following sections describe these models and provide a detailed discussion of how the watercolor program is structured. The structure is broken down into three major sections: Graphical User Interface (GUI), Fluid Simulation, and Watercolor. The GUI section discusses the interfaces that are used and the motivation for primarily using a proprietary interface. Fig. 9 shows the

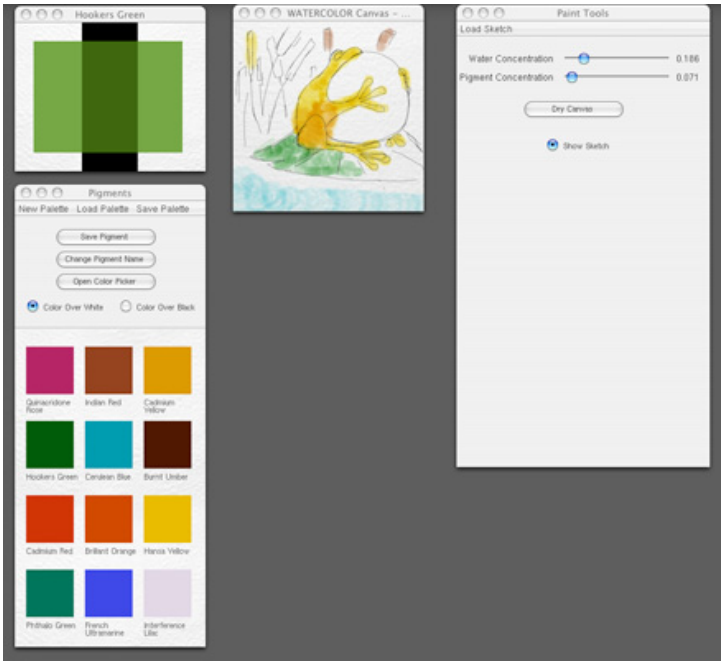

Fig. 9: The Watercolor Program

program's interface. Additionally the palette, brush, and canvas interfaces are discussed. In the fluid simulation section, fluid movement using the lattice Boltzmann method is explained. The section gives a detailed explanation of how the method works on the GPU. Finally, the watercolor section explains the brush, paper and pigment models. This section also describes how all the models interact with each other to recreate watercolor.

A. Graphical User Interface

The watercolor tool uses two different Graphical User Interfaces for interacting with the program. There are several choices when choosing a GUI, however the program uses GLUI [54], a free GUI for OpenGL, and GLUE, a proprietary GUI. These GUIs are used for simplicity and their ability to integrate with OpenGL's shading language GLSL. GLUE is the main interface used and is a custom interface that provides components not available in GLUI. The main component GLUE provides is a color picker. A color picker requires several interface components such as buttons to allow user interaction. While GLUI does have buttons, it is not easy to change their appearance or behavior to work with GLUE. The lack of customization in GLUI requires GLUE provide buttons, radio buttons, sliders and menus so that the color picker will work. GLUE is similar in its appearance to Apple OSX 10.4 tools as seen in Fig. 10. GLUE uses GLUT [36] functions for window management. The color picker shown in Fig. 10 is similar in functionality and layout to Photoshop CS3's color picker tool and uses its design as a reference. Fig. 8 also demonstrates GLUE buttons and sliders. Sliders let the user pick

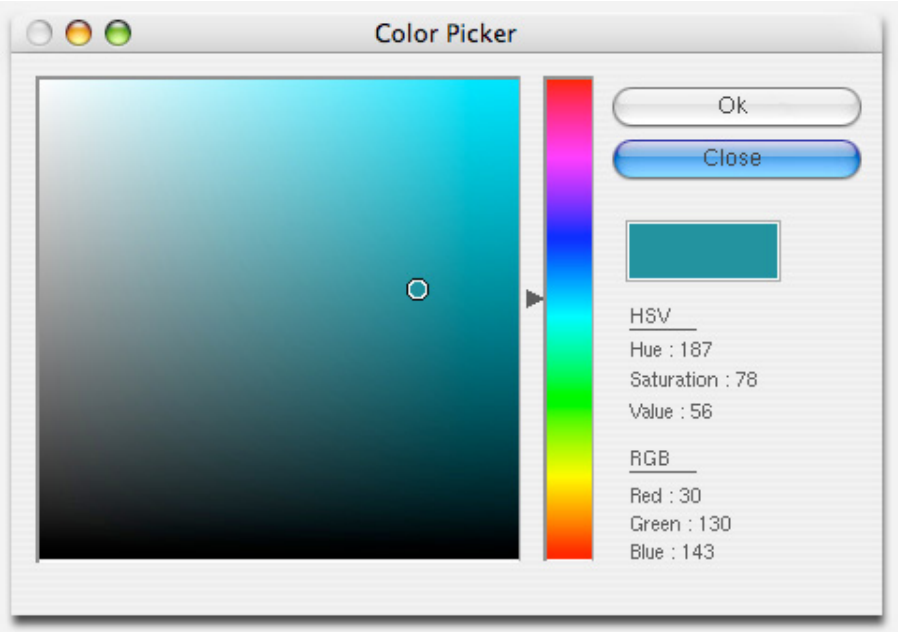

Fig. 10: GLUE Color Picker

the current color and let the user know what the current color is in terms of hue, saturation and value. If the user selects the cancel button, the current color will remain the color at the time the window opened, ignoring all changes. Fig. 10 shows both button states: blue when the mouse clicks or is over the button and white when the mouse is not over the button. The final component of GLUE is a radio button. Radio buttons are blue when selected and white if not selected. While GLUE provides most of the interface components, GLUI is used to provide interface components too time-consuming to implement using GLUE. The program uses GLUI's text box input to capture filenames for saving and loading files. The next sub-sections describe the palette and brush interfaces.

1. Palette

The watercolor tool provides a palette interface so an artist can create, modify, and select pigments. The palette consists of three windows as shown in Fig. 11. The palette uses GLUE for the entire interface, except for palette loading and saving. The left window is the primary interface for managing a pigment's name and color. A palette can hold up to 12 pigments. The pigments are displayed using the Kubelka-Munk Model. The program loads a default palette with 12 common watercolor pigments based on K and S values defined in Curtis et al [3]. However, the user is not limited to the 12 provided pigments. Clicking on a pigment loads the pigment into the top right window in Fig. 11 where it can then be modified using a color wheel. As in Curtis, a pigment is defined by two RGB colors, R_w and R_b, which represent the pigment's appearance in "unit thickness" over white and black. Two radio buttons allow the user to choose whether they are modifying R_w or R_b. Modified pigments can be saved back to the palette using the save pigment button. A palette menu in the left window of Fig. 11 allows users to create a blank palette, load a palette from file, and save a palette to file. The bottom right window in Fig. 9 opens when the load or save palette button is clicked and makes use of the GLUI text box to capture filenames. It also uses GLUI buttons, because GLUE buttons will not integrate with the GLUI textbox.

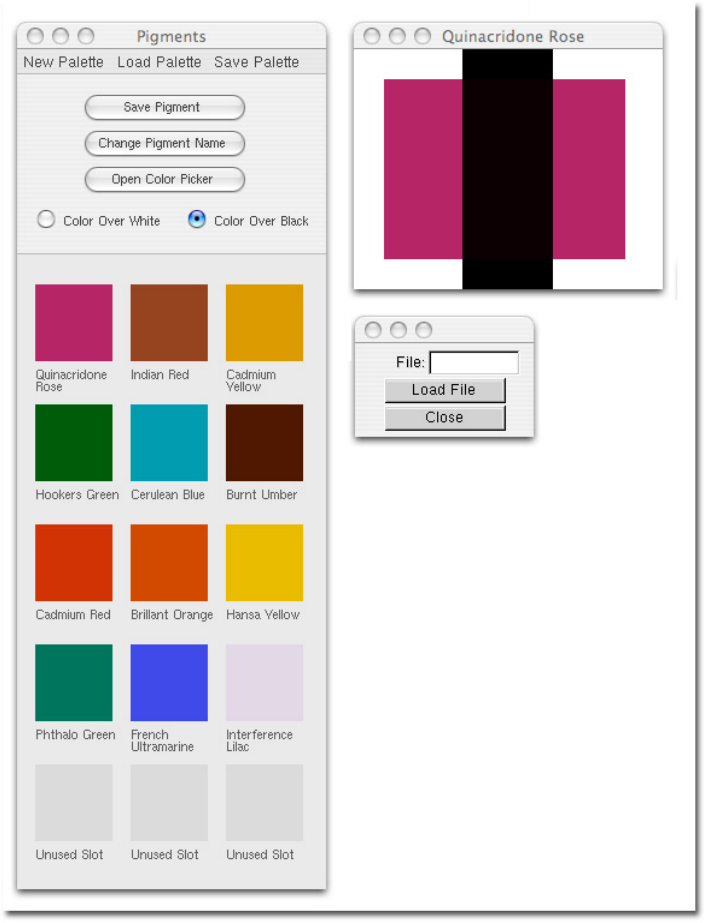

Fig. 11: Palette Interface

2. Brush

The brush interface consists of a visual representation and movement. Visually the brush is represented as a black circle outline and the size is changed using keyboard shortcuts. There are two main choices for controlling brush movement on the canvas. The simplest form of input is a mouse. However a mouse only provides a 2D position on screen. Chu and Tai [4] and Baxter et al. [29] both use a physical pen and tablet for input, which offers more control than a mouse. Their systems can capture brush tilt, pressure, and

position. A disadvantage of using a pen and tablet is the program must use an application programming interface (API) to obtain data from the hardware which can be difficult to implement. While a pen and tablet would provide more control, the brush interface uses a mouse for its simplicity.

3. Canvas

The canvas interface provides controls for the brush and simulation. Fig. 9 shows the canvas interface in the right side of the Fig.. The interface uses 2 sliders, which allow the artist to adjust the pigment and water concentrations in the brush. A button allows the artist to instantly dry the canvas. Often artists first sketch a painting first in order to get proportions and layout correct. A menu and radio button provides the option to load a sketch onto the canvas. The menu opens a file open dialog using GLUI, and then overlays the loaded sketch on the canvas. A radio button toggles the sketch's visibility on and off.

B. Fluid Simulation

A common approach to fluid dynamics is to solve the Navier-Stokes equations [5]. Curtis et al. [3] and Laerhoven et al. [11] use this approach in their watercolor tools. However, as discussed by Chu and Tai [4], the lattice Boltzmann method seems to be a better choice because operations are local and simple, it does not need to solve Poisson equations, and it is easy to incorporate physics that are hard to describe macroscopically.

The watercolor program uses the lattice Boltzmann method for its simplicity and efficient mapping to the GPU.

The lattice-Boltzmann method requires the program to keep track of 9 particle distribution functions f_i, density ρ, and velocity v for each cell. There are 3 textures for the 9 particle distribution functions. Fig. 12 shows the relationship of the particle distribution functions and the texture data. The f_i are grouped according to direction. This is an arbitrary choice, as the f_i could be grouped differently. The velocity and density are stored in a texture with the x and y components in the red and green channels and the density in the blue channel.

A basic lattice-Boltzmann method implementation consists of four sets of operations: streaming, velocity and density computation, boundary detection, and collision. The operations are implemented as fragment shaders. The lattice Boltzmann method requires interfacial boundary conditions to determine the distribution functions

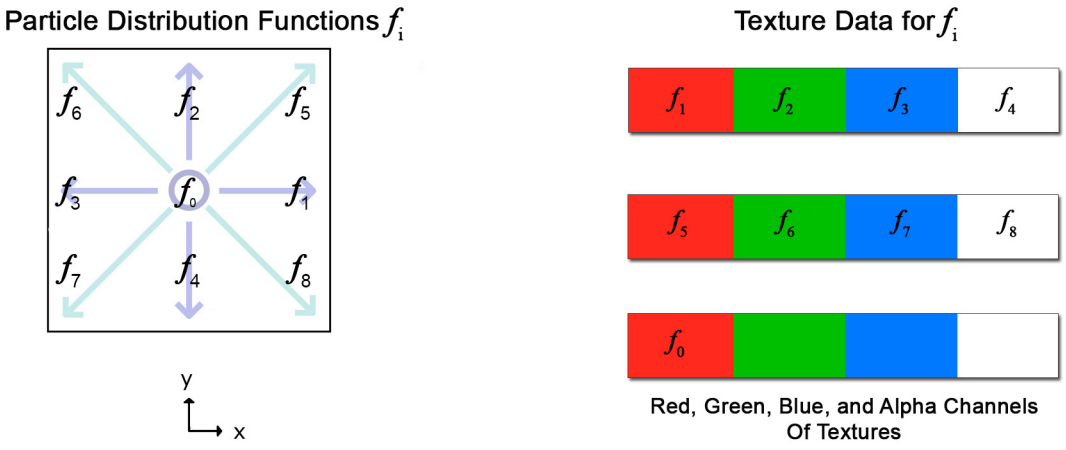

Fig. 12: Particle Distribution Function Texture Storage

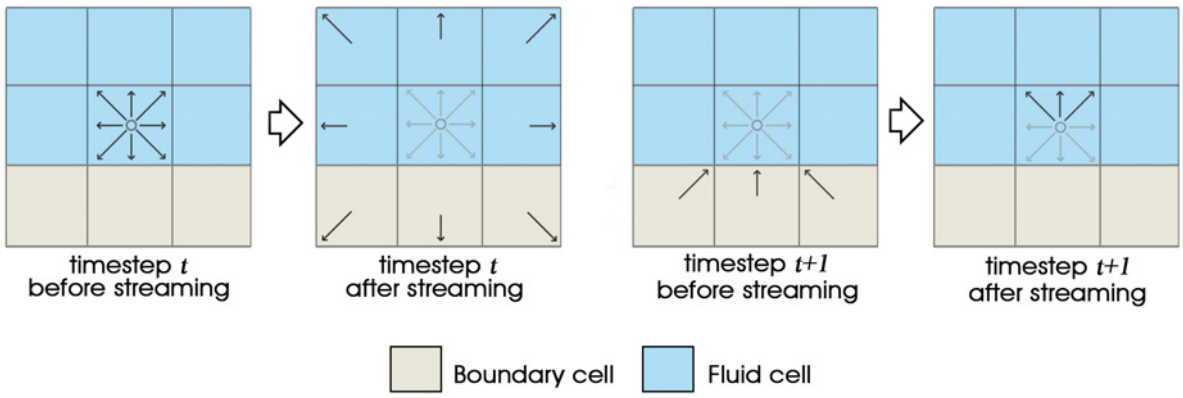

Fig. 13: Half-way Bounce-back Boundary Conditions

at boundary sites. This is accomplished using the half-way bounce-back scheme [5] depicted in Fig. 13. The half-way bounce-back scheme works by reflecting particle distribution functions that enter the boundary cells back in the opposite direction. For example, in Fig. 13, particle distribution functions f_7, f_4, and f_8 stream into the boundary cell in the current timestep. The particle distribution functions are then reversed and will stream back into the cell they came from in the next timestep. On the GPU, the bounce-back step equates to swapping texture channels. So for distribution functions 1-4, the red (f_1) and blue (f_3) channels are swapped and the green (f_2) and alpha (f_4) channels are swapped. Next the streaming step occurs. On the GPU, this is accomplished by swapping channels in a texture, as indicated in Fig. 14. For example, to stream f_1 (the red arrow in Fig. 14) the following GLSL code is used

```
(gl_TexCoord[0].st + vec2(-1.0, 0.0)).r.
```

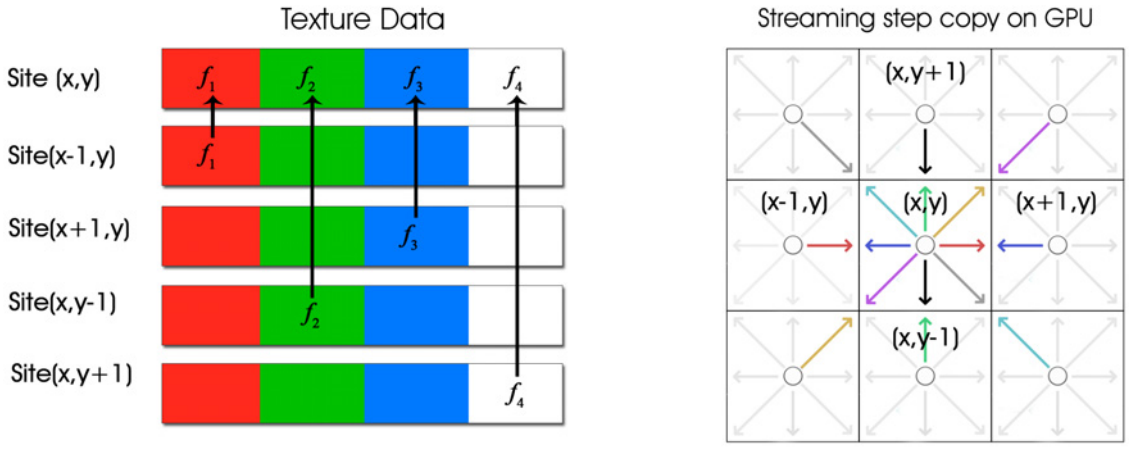

Fig. 14: Streaming Step on the GPU

distribution functions can be written out from a shader to a texture. Since f_0 is stationary, we do not need to stream it. After streaming, the velocity and density shader calculates the new velocity and density based off the new streamed distribution functions. Velocity is calculated using $\mathrm{u} = \frac{1}{\rho_0}\sum_{i=1}^{8} \mathrm{e_i} f_i$ which is equation 2 from Chapter II. The density is given as $\rho = \sum_{i=0}^{8} f_i$. This is equation 1 from Chapter II. The final step in the fluid simulation is calculating the new distribution functions after collision. The collision shader uses the incompressible variant of the LBM,

$$f_i^{(eq)} = w_i\left(\rho + \rho_0\left(\frac{3}{c^2}\mathrm{e_i}\cdot\mathrm{u} + \frac{9}{2c^4}(\mathrm{e_i}\cdot\mathrm{u})^2 - \frac{3}{2c^2}\mathrm{u}\cdot\mathrm{u}\right)\right),$$

which is equation 4 from Chapter II. The shader first calculates the equilibrium distribution function, then uses $f_i' = (1-\omega)f_i + \omega f_i^{(0)}$, which is equation 5 from Chapter II, to find the new distribution functions. Calculating the new f_i is easy on the GPU using

linear interpolation. Finally the shader saves the new f_i to the texture for use in the next timestep. As with streaming, several shaders are needed to save all the distribution functions.

C. Watercolor

The watercolor model consists of a brush model, paper model, fluid simulation, and pigment model. The following sub sections discuss the individual models and how they interact to create watercolor.

1. Brush

Like Curtis et al. [3], the program uses a circle to represent the brush. However, Chu and Tai [4] and Baxter et al. [29] show a more complex brush model can create more realistic paintings. By modeling brush bristles and their interaction with the paper, Chu and Tai are able to get brush strokes that mirror real-life brush strokes. While their model is a better method, it is also difficult to implement and beyond the scope of this thesis. Following Curtis, the brush is a circle, and the footprint is defined as any pixel inside the circle. Pixels only partially covered by the circle are not considered part of the footprint. A bounding box around the circle determines which pixels are in the footprint. A simple test comparing the radius of the circle to all pixels in the bounding box quickly determines which pixels are in the footprint. In addition to the footprint, the program also calculates how fast the brush is moving across the paper surface. Speed is based on the distance traveled from the last brush footprint to the current brush footprint. First the

vector between the strokes is found. Then the magnitude of the vector is scaled to between [0.005, 0.01]. Scaling occurs because the fluid simulation requires small changes in velocity. Consequently, fast movements in the brush will break the simulation. Values between [0.005, 0.01] seem to work well in keeping the velocity and flow speed low. After calculating the speed and footprint, the data is passed to the fluid simulation.

2. Paper

The paper model consists of both a visual and a conceptual representation. Visually the paper is represented using a texture. Curtis et al.[3] suggest noise [38, 39] works well in re-creating watercolor paper textures. The texture is created by first generating an image using Perlin Noise [38]. The image is then applied as a bump map in Autodesk Maya [40].This is rendered in Maya and the resulting image is used as a texture to visually represent the paper. The conceptual representation is an abstraction of how the paper, pigment, and water interact with each other. The paper model is divided into three layers as shown Fig. 15. Like Chu and Tai [4], the paper contains a surface layer, flow layer, and fixture layer. The surface layer is where water and pigment are first deposited. The layer acts as a reservoir supplying water and pigment to the flow layer over time until all pigment and water are depleted. The flow layer advects water and pigment through the paper. Advection in fluid dynamics is a term for describing the transport or movement of material due to the velocity. Pigment is deposited into the fixture layer slowly until all water has evaporated. Some pigment in the fixture layer is absorbed back

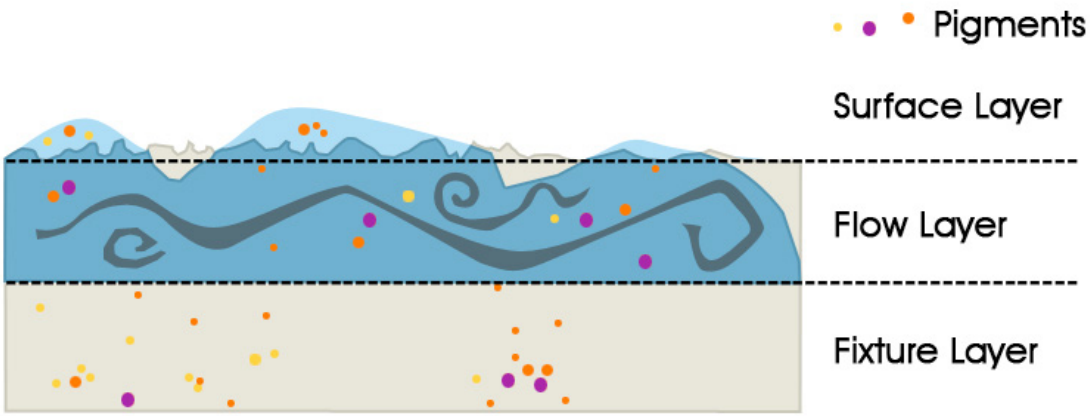

Fig. 15: Paper Layers

into the flow layer and re-deposited later. The paper layers are implemented as sets of shaders, which are described in further detail in the next two sections.

3. Fluid Simulation

Several modifications are made to the LBM to including variable permeability, evolving boundaries, and evaporation. These modifications are important in capturing the behavior of watercolor. As Chu and Tai [4] mention, variable permeability and evolving boundaries are responsible for creating interesting flow patterns. Variable permeability is implemented by having a blocking factor at each site. The blocking factor is used to create a partial bounce-back of distribution functions during streaming. Evolving boundaries are handled by making sites with zero density boundaries. The boundary sites fully bounce-back all streaming distribution functions. Finally density is evaporated over

time by reducing distribution functions during streaming. Together these three modifications help to create edge darkening and flow patterns.

In addition to tracking the state of the f_i, ρ, and v for each cell, the modified lattice-Boltzmann method must also track the amount of water transferred from the surface to flow layer wf, the amount of water in the surface layer ws, the blocking factor κ at each cell for variable permeability, and the height field of the paper h. These variables are stored in textures as shown in Fig. 16. In the modified LBM, the fluid density represents the amount of water in a cell in the flow layer. The height field is generated using Perlin Noise and scaled to the range [0,1]. To limit the number of textures used, the amount of water transferred to the flow layer wf is stored in the alpha channel of the velocity and density texture. The blocking factor κ, amount of water in the surface layer ws, and the height field of the paper h are stored with f_0.

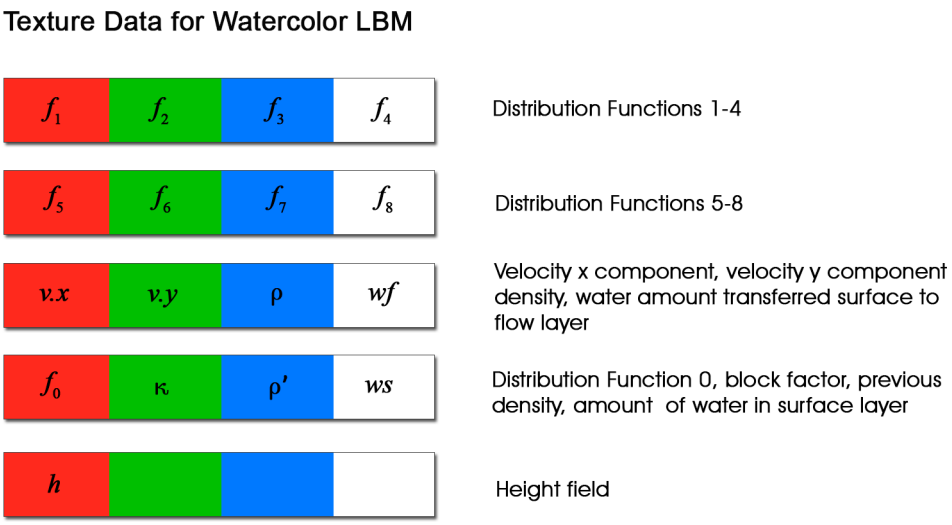

Fig. 16: Texture Storage for Watercolor LBM

The main structure of the lattice-Boltzmann method remains the same. All four operations are performed, but with modifications. First the boundary shader is updated to handle evolving boundaries. The boundary shader is now responsible for setting the blocking factor and the new water amount in the surface layer. A boundary is formed when a cell with no water ($\rho = 0$) is surrounded by cells whose amount of water is less than some threshold η. In this case the boundary site's blocking factor is set to infinity. When any of the dry cell's neighbors' density rises above threshold η, the dry cell's blocking factor is reset to the height field. Additionally non-boundary sites' blocking factors are set to the height field. Finally the water on the surface, *ws*, is updated to max(*ws* – *wf*, 0). Next the streaming shaders stream all f_i with bounce-back and lower the density at boundaries. Fig. 17 describes bounce back for fluid and boundary cells. At site *x*, the blocking factor κ_i is averaged with the blocking factor κ_{i-e_i} to give κ_a. Streaming for both boundaries and fluid cells is described as

$$f_i(x, t+1) = \kappa_a(x) f_i(x, t) + (1 - \kappa_a(x)) f_j(x - e_i, t), \qquad (11)$$

where f_j is the distribution function pointing in the opposite direction of f_i. Fig. 17 shows that the full bounce-back equates to streaming distribution functions back in the opposite direction. Evaporation at boundaries is handled by only evaporating when $\kappa_a = 1$. When this is true a small amount is taken away from the newly streamed distribution functions. In GLSL this is done using $Max(f_i' - b\varepsilon_b, 0)$ where f_i' is the new f_i streamed, ε_b is the evaporation rate specified by the user, and *b* is a Boolean flag

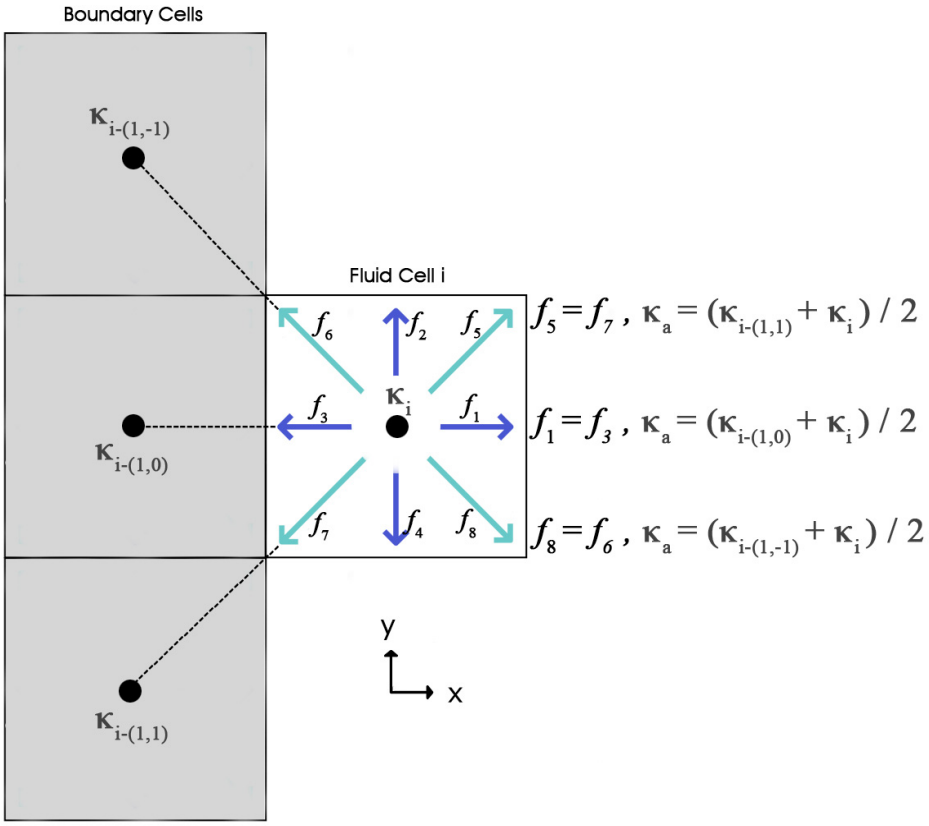

Fig. 17: Streaming with Bounce Back

indicating if the cell is a boundary. The program still uses two shaders for streaming. The velocity calculation remains unchanged, however the density calculation is modified. The density is calculated normally using equation 1 from Chapter II. Next the shader calculates how much water has evaporated from the flow layer using $Max(\rho - \varepsilon_s, 0)$, where ε_s is the evaporation rate for water. Next *wf*, the amount of water transferred from the surface to flow layer, is found using $Clamp\left(ws, 0, Max\left(\beta - \rho, 0\right)\right)$. *Clamp*(x, min, max) is a function that clamps a value x between two numbers min and max. If x > max, x is set to max. If x < min, x is set to min. The value x is left alone if it

is between min and max. β is how much water the fibers in the paper can hold. In the simulation this is set to 1. The final value for the density is $\rho + wf$. The only change made to the collision shader is to add a variable α. The LBM was designed to fill the entire simulation domain [4]. By letting the fluid density represent water in a cell, there will be some cells with no density. As a result there will be cells with negative densities, because the advection built into the LBM carries density away from sites with near zero density. Recall from Chapter II.C, the term $\left(\frac{3}{c^2}\mathrm{e}_\mathrm{i}\cdot\mathrm{u}+\frac{9}{2c^4}(\mathrm{e}_\mathrm{i}\cdot\mathrm{u})^2-\frac{3}{2c^2}\mathrm{u}\cdot\mathrm{u}\right)$ is responsible for advection in the LBM. Chu and Tai [4] suggest adding a parameter α to the term to reduce advection in areas with low densities. The new equilibrium distribution function is

$$f_i^{(eq)} = w_i\left(\rho+\rho_0\alpha\left(\frac{3}{c^2}\mathrm{e}_\mathrm{i}\cdot\mathrm{u}+\frac{9}{2c^4}(\mathrm{e}_\mathrm{i}\cdot\mathrm{u})^2-\frac{3}{2c^2}\mathrm{u}\cdot\mathrm{u}\right)\right). \qquad (13)$$

The variable α is defined by $Smoothstep(0,\lambda,\rho)$ where λ is user specified. $Smoothstep(e_0,e_1,x)$ is a function that provides a smooth transition between edge e_0 and edge e_1 based on the value of x. In *Smoothstep()*, $x = 0$ when x is $< e_0$, $x = 1$ when x is $> e_1$, and x is smoothly interpolated when $e_0 <= x <= e_1$. Therefore *Smoothstep()* will set α to 0 when there is no water, causing no advection to occur. Otherwise α will be greater than zero allowing partial to full advection. Typically, $0.1 \le \lambda \le 0.6$ works well for watercolor based on experimentation. Next pigment movement through the paper layers is described.

4. Pigment

The pigment model is divided into pigment movement and pigment display. Pigment movement follows the model used by Chu and Tai [4] and is divided into three areas: pigment supply, pigment advection, and pigment fixture. Chu and Tai model eastern ink, which does not produce back-runs and granularity. The pigment movement is modified to handle these two effects. Like Curtis et al. [3], pigment display uses the Kubelka-Munk reflectance model. The model is used for both mixing and glazing. As indicated by Curtis, the Kubelka-Munk model works very well for re-producing watercolor's optical effects.

Watercolor pigments behave differently when they are wet versus when they are dry. Wet pigments can still be moved around the canvas, while it is very difficult to move dry pigments. Typically granulation and back-runs occur only when the paper is wet. Therefore the pigment model makes a distinction between wet and dry pigments to allow the artist full control over the painting. Wet and dry pigments are stored in different layers. The layers are implemented as RGBA textures. Dry pigments can be

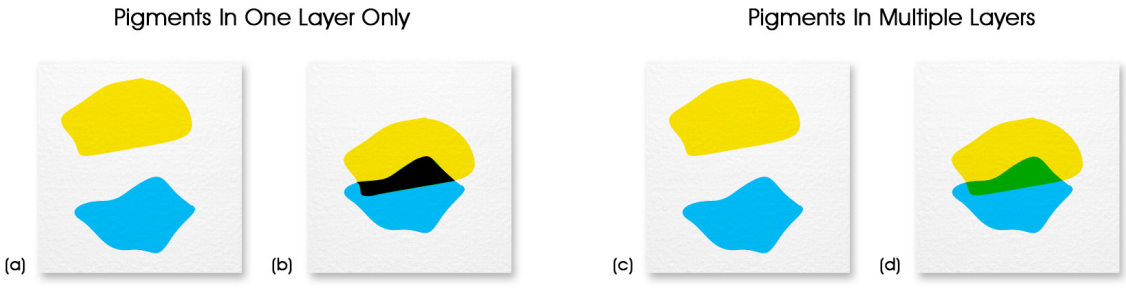

Fig. 18: Pigment Layers

stored in one layer because dry pigments cannot be moved. Wet pigments however, require multiple wet layers because they are tracked as a group and adding more water may move them. Pigments are tracked as a group, and not individually, because it is computationally efficient. However, this presents some problems as Fig. 18 illustrates. If only one layer is used to track pigments, a problem occurs when two pigments of different color overlap (Fig. 18b). Recall the Kubelka-Munk model optically mixes color based on two layers' light scattering and absorption properties. The model does not know how to optically mix the color of two different pigments if they are not in different layers. The problem is fixed by separating pigments into different layers according to color (Fig. 18d). Next a detailed description is given on the pigment movement in the wet and dry layers.

Pigment movement tracks the concentration of pigments in the different paper layers. Pigment concentrations in the supply, flow, and fixture layers are denoted as P_s, P_f, and P_x respectively. Concentrations are stored in the red, green, and blue channels of a texture as indicated in Fig. 19. As mentioned previously, the program stores pigment

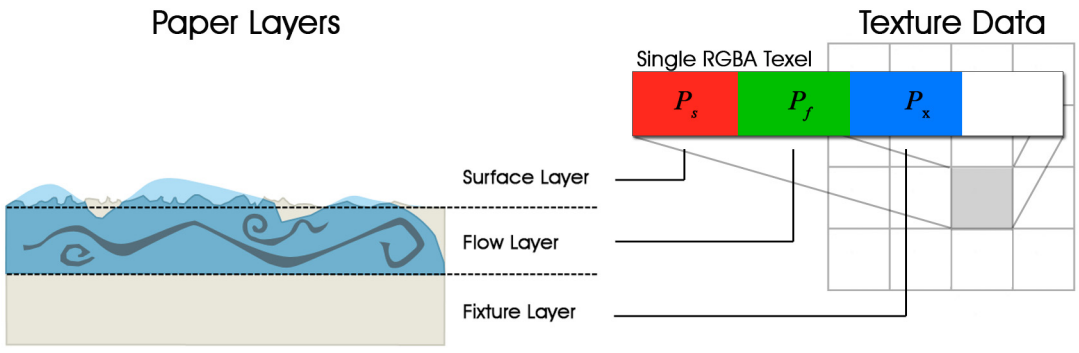

Fig. 19: Pigment Concentration Storage

concentrations according to their color. Hence a texture is required for every different pigment color. Three shaders model the movement of pigment through the paper layers as depicted in Fig. 20. Pigment is first deposited into the surface layer. The amount of pigment transferred from the surface to flow layer is determined by a ratio of the amount of water in the flow layer to the amount of water being transferred to the flow layer. Specifically,

$$P_f = \frac{P_f \rho + P_s wf}{\rho + wf} \,. \tag{14}$$

After P_f is updated, the shaders advect pigment through the flow layer. There are two types of cells, cells that are already wet and cells that are becoming wet. The new pigment concentration at site x for cells already wet is found by tracing the velocity backwards. In this case

$$P_f^*(\mathrm{x}) = P_f(\mathrm{x} - \mathrm{u}(\mathrm{x})). \tag{15}$$

The new pigment concentration at site x for cells becoming wet is given as

$$P_f^*(\mathrm{x}) = \frac{1}{\rho}\sum_{\mathrm{i}=1}^{8} f_{\mathrm{i}} P_f(\mathrm{x} - \mathrm{e}_{\mathrm{i}}). \tag{16}$$

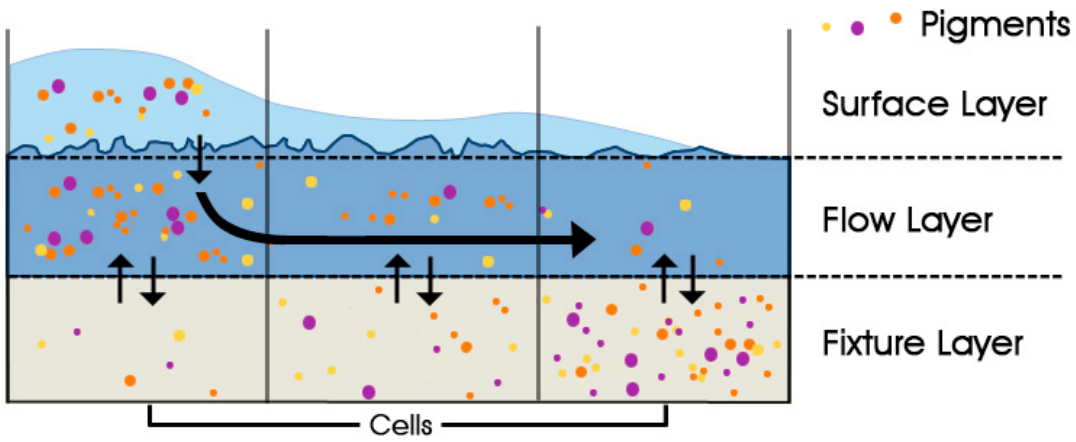

Fig. 20: Pigment Movement in Paper Layers

As pigments move through the flow layer, they are slowly deposited into the fixture layer. As Fig. 20 indicates, the pigment model allows pigment to deposit more when the paper is drier and less when wet. This is accomplished by basing pigment deposition on the amount of water at a site. The concentration in the fixture layer is updated using the following process. First the shader calculates the amount of water evaporated since the last time step. This quantity is denoted *wl* and it equals $\rho' - \rho$, where ρ' is the density in the last time step. The next step finds ψ, the percent of water lost, using $\frac{wl}{\rho'}$. After finding ψ, the shader determines how much pigment to deposit based on the amount of water evaporated and density. The equation is

$$P_{fix} = Max\left(\psi\left(1 - Smoothstep(0,\zeta,\rho)\right),\phi\right). \qquad (17)$$

P_{fix} is the amount of pigment to deposit, ζ modulates P_{fix} by dryness and ϕ is a base rate of deposition.

Granularity occurs when pigments clump together in the valleys of the paper and is created using

$$\chi\left(1 - Smoothstep(0, \mu, h)\right). \quad (18)$$

Since granularity happens most in deep valleys of the paper, the equation only lets deep areas receive pigment. In the equation, χ is a weight for the strength of the granularity, μ is a threshold value for which granularity occurs and *h* is the height field of the paper. The shader only allows granularity to occur when the velocity's magnitude is below a user specified rate. At higher velocities the flow speed will be greater and it is less likely that pigments will settle. To account for back-runs, the shader checks if the velocity's magnitude is greater than τ, the settling speed of the pigments. Back-runs occur when water flows back into a damp region creating severe branching patterns. Hence, the water flow must be high enough to lift pigments from the paper and redeposit them. For damp regions, when the magnitude is greater than τ, P_{fix} is set to $P_{fix} - (\theta \rho P_x)$. This removes pigment from the fixture layer and re-deposits it into the flow layer. θ is a parameter for controlling the amount re-absorbed into the flow layer. Finally the shader updates P_f to $P_f - P_{fix}$ and P_x to $P_x + P_{fix}$. In the case of back-runs, the signs are reversed. The pigment movement shaders are run for each texture in the wet layer. The program does not save pigment concentrations in the dry layer because they will not ever be moved. Rather the dry layer only saves reflectance information. When the user presses the "Dry Paper" button in the canvas interface, the pigments in the wet layers are composited together with the dry layer. Then the wet layers' pigment concentrations are set to zero. Next the compositing process is discussed.

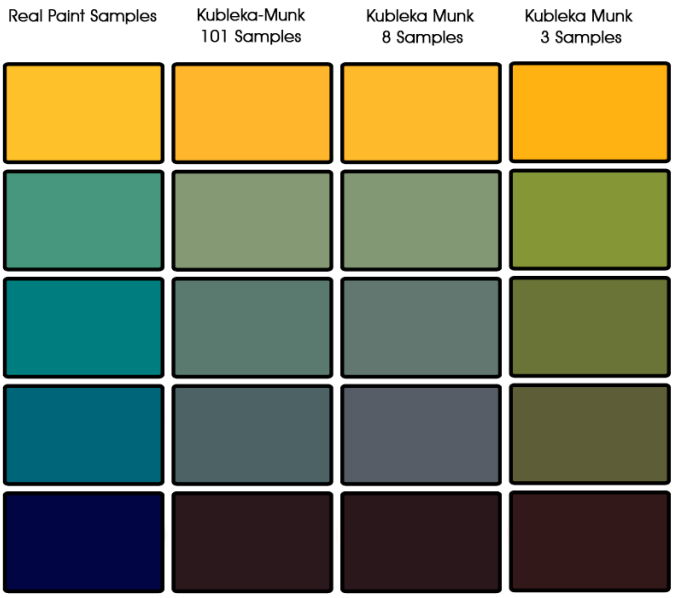

Fig. 21: Comparison of Kubelka-Munk Samples based on Baxter et al. [29]

Pigments are displayed using the Kubelka-Munk reflectance model. The model is implemented using both the GPU and the CPU. The CPU calculates the absorption coefficient *K* and scattering coefficient *S* and the GPU produces the total reflectance *R*. Similar to Curtis et al. [3], the program uses a three-wavelength representation for the Kubelka-Munk model. This works well since the three wavelengths map to the red, green, and blue channels of a texture. Additional wavelengths add more accuracy as demonstrated in Fig. 21, but require special equipment to capture the spectral measurements. This equipment is not readily available, thus the program follows Curtis's method of setting *K* and *S* interactively. After the user specifies R_w and R_b using the interface in Fig. 11, the program derives the absorption and scattering coefficients on the CPU. The GPU shader then finds the total reflectance of the pigments. The shader

takes the total reflectance *R* for the bottom layer and the absorption and scattering coefficients for the top layer as input. Then the top layer's reflectance and transmittance are found and composited optically with the bottom layer producing a total reflectance for both layers. Since three wavelengths are used, the total reflectance becomes the RGB color of the pigments that is displayed on the screen. The shader must be run every timestep for both the dry and wet layers. The compositing of these layers is an iterative process. Starting with the dry layer, the layers are composited from bottom to top. The final reflectance from compositing two layers is used as the bottom layer's reflectance in the next iteration of the shader. Pigment display uses only the diffuse reflectance from the Kubelka-Munk model. Since watercolor is fairly diffuse, there is no need for specular reflection.

CHAPTER IV

EVALUATION

The following evaluation is divided into two sections, the visual results and the program's performance. The visual output section discusses how well paintings produced using the digital watercolor tool match real-life watercolor paintings. The performance section evaluates the speed and simulation size of the system.

A. Visual Output

Both watercolor effects and paintings produced using the system are examined in evaluating the visual output from the watercolor program. The watercolor tool simulates a variety of watercolor effects illustrated in Fig. 22. Comparing the results in Fig. 22, to their real-life counterparts in Fig. 23, shows the tool does a good job of re-producing watercolor effects. Like real watercolor (Fig. 23e), the tool accurately models flow effects (Fig. 22b) when a large amount of water is deposited on the paper. This tool handles this effect particularly well, betraying no signs of its digital origins. The tool also does a good job handling the edge darkening effect (Fig. 23b) as seen in Fig. 22b. As with real watercolor, the program allows pigments to deposit more when the paper is drier. The watercolor tool shows some weakness in reproducing backruns (Fig. 23c). As Fig. 22a shows, the backruns do form, but the severe branching is not as strong as in real watercolor. In this case the height field used to represent the papers fibers shows its limitations. This effect might be improved by using

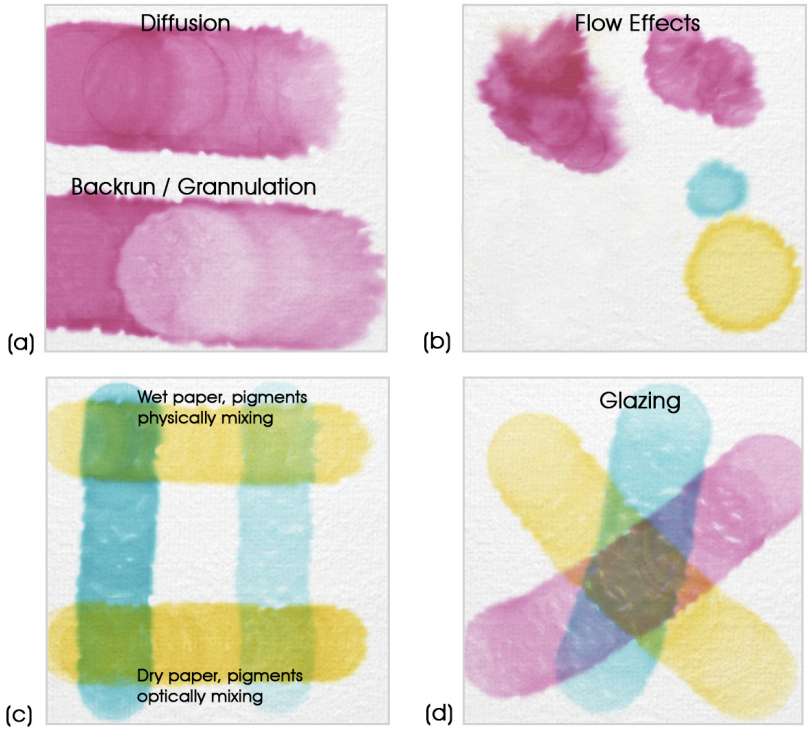

Fig. 22: Digital Watercolor Effects

a different more complex paper model. The tool does a moderately good job at reproducing granulation (Fig. 23d) as exhibited in Fig. 22a. Again, this effect would benefit from a more complex paper model that better describes the intricate

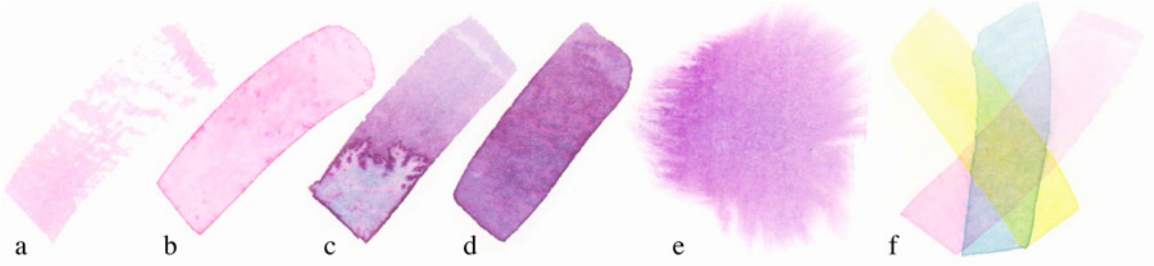

Fig. 23: Real Watercolor Effects from Curtis et al. [3]. (a) dry-brush (b) edge darkening (c) back-runs (d) granulation (e) flow patterns (f) glazing

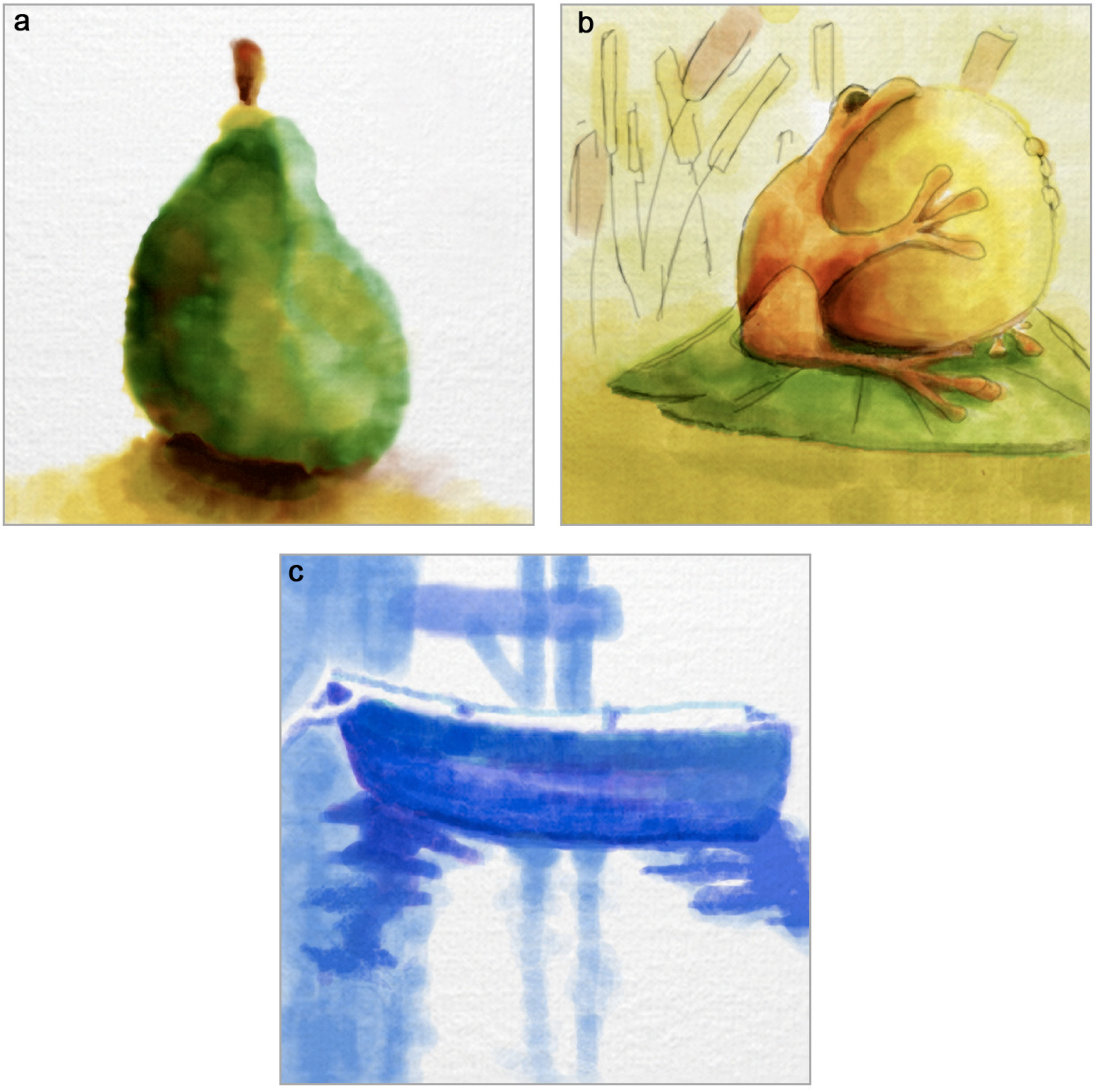

Fig. 24: Paintings From the Digital Watercolor Tool

relation between fibers and air in the paper. The tool only handles wet-in-wet and wet-on-dry techniques. Dry-brush techniques were not attempted due to time constraints.

The program's use of the Kubelka-Munk reflectance model produces convincing results. As illustrated in Fig. 22d, the model helps in faithfully reproducing glazing (Fig. 23f). The Kubelka-Munk model also does a good job at pigment mixing. Fig.

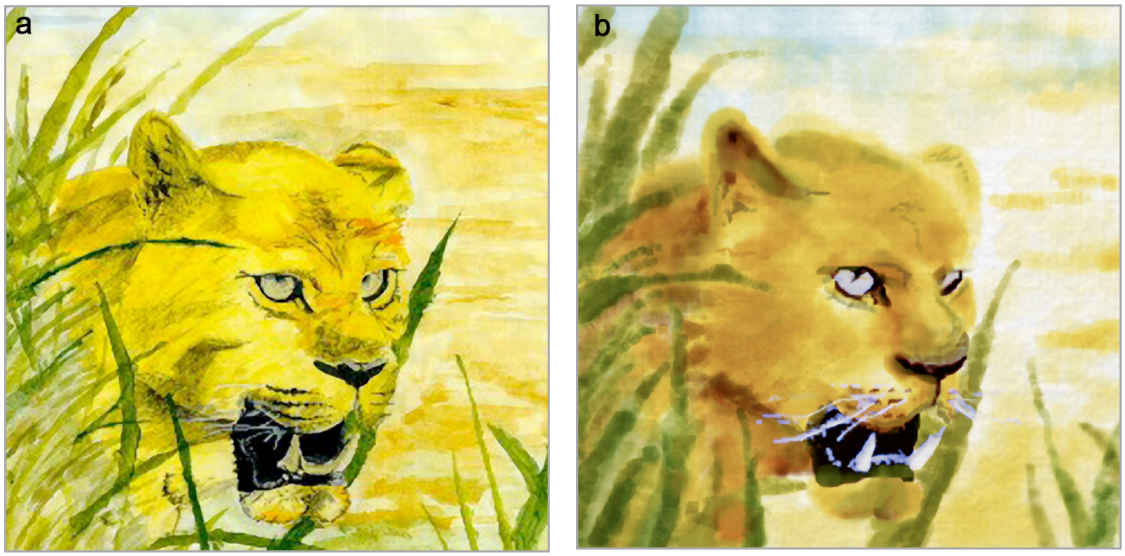

Fig. 25: Real Vs. Digital. (a) Real Watercolor (b) Digital Watercolor

22c shows the difference between physical and optical mixing. In the top of the Fig., pigments mix together since the paper is still wet. This produces a cool green. However in the bottom, where pigments mix optically, a warmer green is produced. Overall the tool re-produces effects created by real watercolor well.

Fig. 24 shows several paintings produced by the program. Fig. 24b makes use of the sketch-loading feature to aid in the painting. A comparison of a real watercolor painting (Fig. 25a) and a digital watercolor painting from the program (Fig. 25b) show the digital watercolor closely resembles the real watercolor painting. It becomes apparent from the comparison that having a better brush model and larger canvas would make adding detail easier. The digital watercolor painting can't capture some of the small detailed brushstrokes and can't vary the width of a stroke due to the program's simple brush model. Fig. 26 shows an automated painting. In this painting

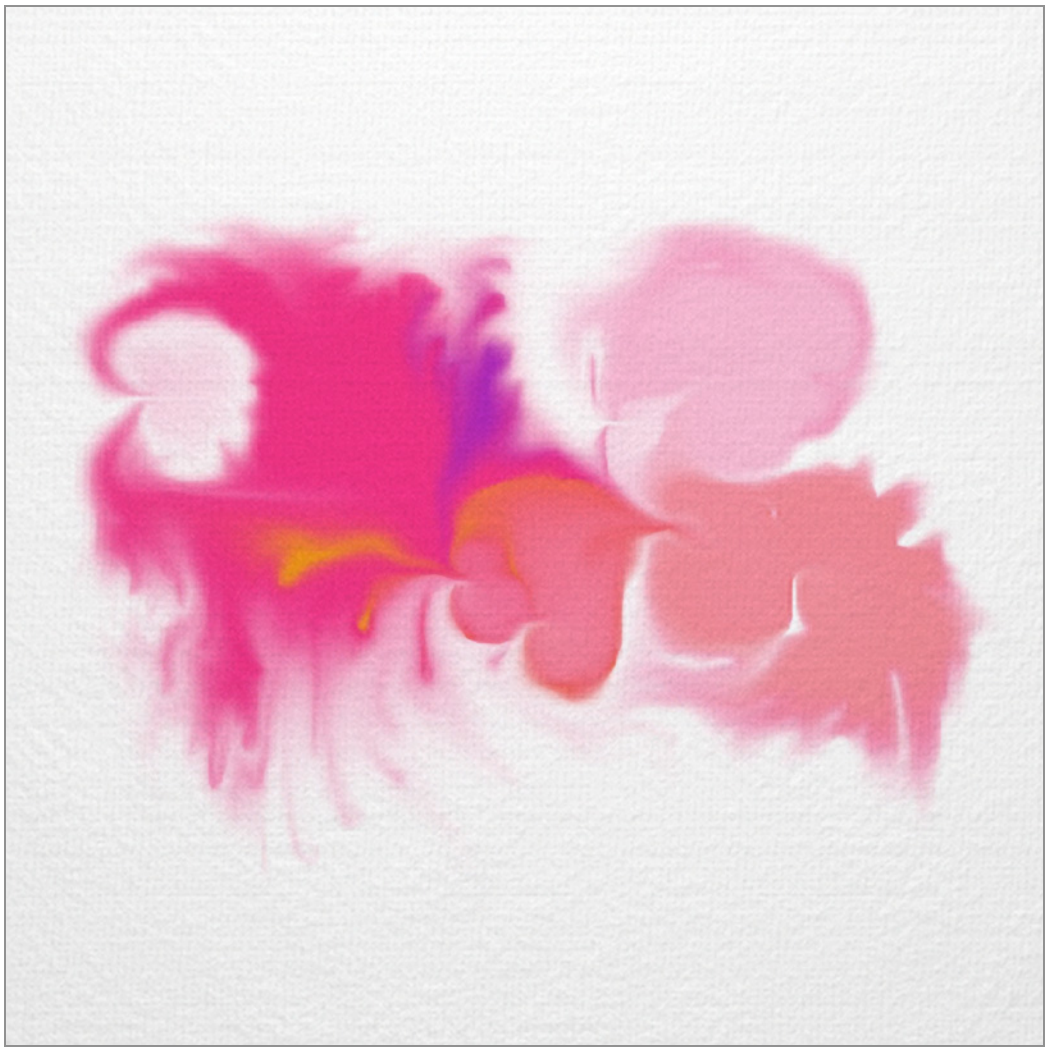

Fig. 26: An Automated Digital Watercolor Painting

no brush strokes were created, instead the painting was driven by cells with a constant acceleration. The results support the finding that the brush is the limiting factor in producing small detailed paintings. This painting has a lot of small detail created by the fluid simulation running through the peaks and valleys of the paper.

B. Performance

One of the goals of the thesis was to create real-time interaction between the program and artist. Real-time interaction begins at around 6 frames per second (fps) [41]. The digital watercolor tool runs at approximately 47 fps on a 256x256 simulation grid using an Nvidia Geforce 8600m graphics card. When run on a 512x512 simulation grid, the program ran at 17 fps. While this qualifies as real-time, it did not produce feedback fast enough to paint effectively. The main bottleneck in performance is the need to write a large amount of data to and from the GPU in each timestep. Every timestep, there are 7 shader calls for the lattice Boltzmann method, 25 shader calls for the pigment movement, and 13 calls for the compositing and display. Each shader call passes texture data to the GPU and receives back texture data after execution of the shader code. Eliminating the number of times this is done shows a significant increase in frame rate. The number of shader calls is a limitation of the technology, not the tool itself. As the technology matures, and GPUs can write out to an arbitrary amount of textures, performance will increase.

CHAPTER V

CONCLUSIONS AND FUTURE WORK

A. Conclusions

The goal of this thesis was to create an extensible real-time watercolor tool using a combination of physically-based techniques and the GPU. The images produced by the watercolor program look very similar to traditional watercolor paintings. Additionally the tool behaves like real watercolor in the effects it reproduces. The framework presented in this paper lays a foundation for future work in digital watercolor and demonstrates the effectiveness of both the lattice Bolztmann method and Kubelka Munk Reflectance model using the GPU.

B. Future Work

The work presented here provides several directions for future work. An improved brush model would greatly increase the artist's control when painting. Specifically, a physically based brush model combined with a pen and tablet would enable an artist to paint finer detail and would create more realistic brush strokes than are possible with a mouse. The pigment model would be another area that would be interesting to develop further. Tracking pigments in groups is efficient, however it greatly simplifies pigment characteristics. This simplification eliminates much of the pigments' complexity in terms of their interaction with the paper and water. If pigments could be tracked individually, then each pigment could be given properties such as color, staining power, and density.

This would allow for more realistic pigment behavior. As computational power on the GPU increases and the technology matures, tracking pigments individually should become feasible. A final area of interest is the paper model. While the paper model presented is effective, a more complex model might produce better results. Finding a way to model the intertwining fibers of the paper should lead to more realistic back-runs, diffusion, and granulation.

REFERENCES

1. A. Gooch and B. Gooch, *Non Photorealistic Rendering*. A. K. Peters, 2001.

2. S. Schlechtweg and T. Strothotte, *Non-Photorealistic Computer Graphics: Modeling, Rendering, and Animation*. Morgan Kaufmann Publishers, 2002.

3. C. J. Curtis, S. E. Anderson, J. E. Seims, K. W. Fleischer, and D. H. Salesin, "Computer-Generated Watercolor," *SIGGRAPH '97: Proceedings of the 24th Annual Conference on Computer graphics and Interactive Techniques*, vol. 16, no. 3, pp. 421-430, Aug. 1997.

4. N. S.-H. Chu and C.-L. Tai, "MoXi: Real-Time Ink Dispersion in Absorbent Paper," *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 504-511, Jul. 2005.

5. C. B. Vreugdenhil, *Numerical Methods for Shallow-Water Flow*. Kluwer Academic Publishers, 1994.

6. S. Succi, *The Lattice Boltzmann Equation for Fluid Dynamics and Beyond*. Oxford University Press, 2001.

7. D. Small, "Modeling Watercolor by Simulating Diffusion, Pigment, and Paper Fibers," *Proceedings of SPIE*, vol. 1460, pp. 140-146, Feb. 1991.

8. F. H. Harlow and J. E. Welch, "Numerical Calculations of Time-Dependent Viscous Incompressible Flow of Fluid with Free Surface," *Physics of Fluids*, vol. 8, no. 12, pp. 2182-2189, 1965.

9. N. Foster and D. Metaxas, "Realistic Animation of Liquids," *Graphical Models and Image Processing*, vol. 58, no. 5, pp. 471-483, 1996.

10. P. Kubelka and F. Munk, "Ein Beitrag zur Optik der Farbanstriche," *Zeitschrift fur Technische Physik*, vol. 23, pp. 593-601, 1931.

11. T. V. Laerhoven and F. V. Reeth, "Real-time Simulation of Watery Paint: Natural Phenomena and Special Effects," *Computer Animation and Virtual Worlds*, vol. 16, no. 3-4, pp. 429-439, Jul. 2005.

12. J. Stam, "Stable Fluids," *Proceedings of Siggraph 1999*, pp. 121-128, Aug. 1999.

13. J. Burgess, G. Wyvill and S. A. King, “A System for Real-Time Watercolor Rendering,” *Proceedings of the Computer Graphics International 2005*, pp. 234-240, Jun. 2005.

14. L. Boltzmann, *Lectures on Gas Theory*. Courier Dover Publications, 1995.

15. J. Hardy, O. de Pazzis and Y. Pomeau, “Molecular Dynamics of a Classical Lattice Gas: Transport Properties and Time Correlation Functions,” *Physical Review A*, vol. 13, no. 5, pp .1949-1961, May 1976.

16. U. Frisch, B. Hasslacher and Y. Pomeau, “Lattice Gas Automata for the Navier-Stokes Equations,” *Physics Review Letters*, vol. 56, pp. 1505-1508, Apr. 7, 1986.

17. G. McNamara and G. Zanetti, “Use of the Boltzmann Equation to Simulate Lattice-Gas Automata,” *Physics Review Letters*, vol. 61, pp. 2332-2335, Jul. 29, 1998.

18. F. Higuera and J. Jimenez, “Boltzmann Approach to Lattice Gas Simulations,” *Europhysics Letters*, vol. 9, p. 663, Aug. 1989.

19. F. Higuera, S. Succi and R. Benzi, “Lattice Gas Dynamics with Enhanced Collisions,” *Europhysics Letters*, vol. 9, p. 345, Jun. 1989.

20. Y. Quian, D. d’Humieres and P. Lallemand, “Lattice BGK Models for Navier-Stokes Equation,” *Europhysics Letters*, vol. 17, p. 79, Feb. 1992.

21. X. He and L.-S. Luo, “Lattice Boltzmann Model for the Incompressible Navier-Stokes Equation,” *Journal of Statistical Physics*, vol. 88, nos. 3,4, pp. 927-944, 1997.

22. N. Thurey, “A Single-Phase Free-Surface Lattice Boltzmann Method,” *Masters Thesis*, University of Erlangen-Nuremberg, 2003.

23. M. C. Sukop and D. T. Thorne Jr., *Lattice Boltzmann Modeling An Introduction for Geoscientists and Engineers*. Springer, 2005.

24. C. S. Haase and G. W. Meyer, “Modeling Pigmented Materials for Realistic Image Synthesis,” *ACM Transactions on Graphics*, vol. 11, no. 4, pp. 305–335, Oct. 1992.

25. C. Donner and H. W. Jensen, “Light Diffusion in Multi-Layered Translucent Materials,” *ACM Transactions on Graphics*, vol. 24, no. 3, pp. 1032- 1039, Jul. 2005.

26. F. E. Nicodemus, J. C. Richmond, J. J. Hsia, I. W. Ginsberg and T. Limperis, "Geometrical Considerations and Nomenclature for Reflectance," *National Bureau of Standards*, pp. 3-9, 1977.

27. C. S. Oh and Y.-H. Nam, "GPU-based 3D Oriental Color-Ink Rendering," *ACM International Conference Proceeding Series*, vol. 157, pp. 142-147, Dec. 2005.

28. J. S. Scott, "GPU Programming for Real-Time Watercolor Simulation," Masters Thesis, Dept. Arch., *Texas A&M Univ.*, 2004.

29. W. Baxter, J. Wendt and M. C. Lin, "IMPaSTo: A Realistic, Interactive Model for Paint," *Proceedings of the 3rd International Symposium on Non-Photorealistic Animation and Rendering NPAR '04*, pp. 45-148, Jun. 2004.

30. W. Baxter, Y. Liu and M. C. Lin, "A viscous paint model for interactive applications," *Computer Animation and Virtual Worlds*, vol. 15, no. 3-4, pp. 433-441, Jul. 2004.

31. R. J. Roost, *OpenGL Shading Language Second Edition*. Addison-Wesley, 2006.

32. *General Purpose Calculations on the GPU*. [Online]. Available: http://www.GPGPU.org.

33. M. J. Harris, G. Coombe, T. Scheuermann, and A. Lastra, "Physically-Based Visual Simulation on Graphics Hardware," *SIGGRAPH/EUROGRAPHICS Workshop On Graphics Hardware*, pp. 109-118, Sept. 2002.

34. W. Li, X Wei, and A. Kaufman, "Implementing Lattice Boltzmann Computation on Graphics Hardware," *The Visual Computer*, vol. 19, no.7-8, pp. 444-456, 2003.

35. P. Rademacher. (1999, June 10). *GLUI User Interface Library (2nd ed)*. [Online]. Available: http://glui.sourceforge.net.

36. "GLUT Window System Toolkit," http://www.opengl.org/resources/libraries/glut/. Accessed March 12, 2008.

37. "Adobe Photoshop CS3," http://www.adobe.com. Accessed March 12, 2008.

38. K. Perlin, "An Image Synthesizer," *SIGGRAPH '85: Proceedings of the 12th Annual Conference on Computer graphics and Interactive Techniques*, pp. 287-296, July 1985.

39. S. P. Worley, “A Cellular Texturing Basis Function,” *Siggraph ’96 Proceedings*, pp. 291-294, 1996.

40. “Maya 8.5,” http://usa.autodesk.com. Accessed March 12, 2008.

41. T. Akenine-Moller and E. Haines, *Real-Time Rendering*, A K Peters, 2002.

VITA

Name:	Patrick O'Brien
Email Address:	patrick@patrickobrienart.com
Web Site:	www.patrickobrienart.com
Education:	B.B.A., Management Information Systems with a Minor in Mathematics, University of St. Thomas, 2001
	M.S., Visualization Sciences, Texas A&M University, 2008