Simulating Watercolour Painting

Anders Langlands
NCCA Bournemouth University

Abstract

In this paper I describe the research and development for a real-time watercolour painting program. I present my research into related areas of interest, such as fluid dynamics, cellular automata and Kubelka-Munk theory. I look closely at two existing systems for simulating the properties of watercolour paints, and assess their possible applicability to real-time painting. I go on to develop two programs to test my ideas, and provide the design for a third that should solve the problems encountered with previous implementations.

1. Introduction

In recent years, much attention has turned to reproducing natural-media effects for digital painting. As processing power has advanced, so has the complexity and believability of algorithms used to create these effects.

Watercolour is perhaps the hardest natural medium to simulate. Oil paints and hard media such as pastels and pencil can effectively be described in the immediate contact reaction between the artist's tool and the paper's surface. Watercolour painting, on the other hand, depends heavily upon the motion of the pigment solution applied to the paper. This process continues after the artist's brush has left the paper and, depending on the environment, the system will continue to evolve for some time after the artist has finished painting. It is for this reason that simulating watercolour painting convincingly presents such a challenge.

1.1 Related Work

Small (1991) presents a method for simulating watercolour effects using a system of cellular automata on a Connection Machine. Curtis et al (1997) also look at Small's work and extend Small's system to incorporate the shallow-water equations of fluid dynamics and Kubelka-Munk theory for a more accurate result. The resulting images [Fig.1.1] are pleasing to the eye and exhibit many of the features of real watercolour paintings.

Figure 1.1 *Realistic watercolourisation of a low-resolution video frame by Curtis's system.*

1.2 Overview

In the next section I first present my research into traditional watercolour techniques, the properties of watercolour paints and modelling the optical properties of the paint for computer display using Kubelka-Munk theory. In section 3 I then present my research into cellular automata and fluid dynamics, and how these apply to the simulation of the watercolour medium. In section 4 I make a detailed analysis of the methods described by Small and Curtis et al, and assess their applicability to the problem of interactive simulation of watercolour, I also describe the implementation of a simple painting program in C++ using Small's method as a template. In section 5 I go on to describe my first proposed method using particles, and also describe the partial implementation of this method. In section 6 I make an assessment of the work produced so far, before describing the possible design of a realistic, interactive watercolour painting program in section 7. In section 8 I assess my success in carrying out this project and draw my conclusions.

2. Research

2.1 Traditional watercolour

Watercolour as a medium has been in use for over 30,000 years. Watercolour as we know it today first made its successful transition to paper (as opposed to walls) in the 15th Century, one of the first purveyors being Albrecht Dürer.

Watercolour is best known in the paintings of 18th and 19th Century England, made famous by such artists as Turner and Constable (Parramón, 1993).

San Giorgio Maggiroe from the Customs House by J.M.W. Turner

The Piece of Turf by Albrecht Dürer

The paint itself consists of minute pigment grains held in suspension in a mixture of water and binding agent. The binding agent helps the pigment to adhere to the paper's surface (Curtis et al, 1997).

Curtis et al (1997) state that it is just as important to effectively simulate the effects watercolour produces as it is to simulate its physical properties.

Bolton (2000) and Harnson (2000) list the basic techniques and effects as follows:

- **Wash**: areas of colour applied to dry or wet paper with a well-loaded, large brush. A wash can be flat: a single colour, graded: one colour applied in a smooth blend from dark to light, or variegated: a smooth blend of two or more colours.
- **Wet-into-wet**: paint is applied to a damp or wet surface giving a "ghostly" effect caused by the patterns of the water's flow.
- **Wet-onto-dry**: paint is applied onto dry paper or already dry paint.
- **Glazing**: when thin layers of colour are applied over the top of one another, the result is a rich, translucent tone.
- **Dry-brush**: by blotting the brush to remove most of the water and drawing over the paper at the right angle, colour is deposited only on the high points of the paper's surface creating a rough, textural effect.
- **Granulation** is caused by the pigments settling down into the recesses in the paper's surface. Different paints exhibit more or less of this effect depending on the weight of their pigments.

Curtis et al (1997) also make specific mention of edge-darkening. When wet paint is applied onto a dry surface, evaporation at the edges of the stroke causes the paint to flow outwards, causing more pigment to be deposited at the edges of the stroke. This is also a characteristic feature of watercolour, which a skilled artist knows how to control.

From looking at these effects and their use by professional artists in watercolour paintings, one can break down watercolour into two separate processes: **flow** and **optical composition**. Flow is governed by the dynamics of the water in which the pigment is suspended. The way in which the water moves over the paper is extremely important for all of the effects listed above, and any simulation of watercolour painting must either simulate water flow accurately, or believably fake its end results. Optical composition is the description of how the pigments in the paint are combined whether by mixing or glazing, and how light reflects off them to produce the image in the viewer's eye. In the next section I investigate Kubelka-Munk theory as a method to accurately describe the

optical properties of pigmented materials such as paints. I then go on to investigate fluid dynamics and assess its suitability for describing flow effects in a watercolour painting application.

2.2 Kubelka-Munk Theory

The colour model with which computer graphics artists are most familiar is the additive RGB triplet model, corresponding to the red, green and blue electron guns found in a common television set or computer monitor. This model is in direct contrast to the way in which media such as watercolour, which is nearly transparent, affect light passing through them. Filtering media are subtractive. That is to say that combining a red filter, a blue filter and a green filter will produce black. In contrast, combining red, green and blue in the additive model will produce white.

Subtractive colour can be calculated simply by using the CMY triplet model, which is a linear transformation of the RGB colour space using the formula:

$$\boldsymbol{C}_{CMY} = [1\ 1\ 1] - \boldsymbol{C}_{RGB}$$

The results of any colour model used in a paint application must be transformed back to RGB space for projection on a monitor, so using the CMY model is very attractive for computer graphics applications because of its simplicity.

However, subtractive colour mixing assumes that the material in question is transparent enough that light can pass straight through. For thin washes of watercolour paints, one can often assume that this is the case (Haase and Meyer, 1991), but it will not always be so. This is because suspended particles of pigment will reflect and scatter incoming light in an unpredictable way [see Fig. 2.1 below].

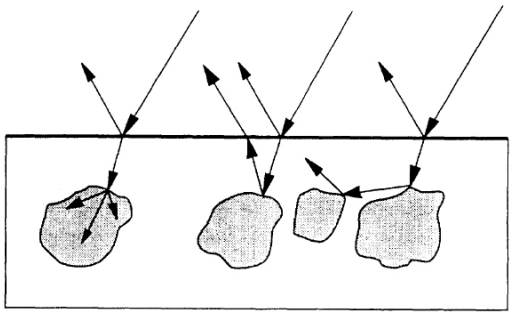

Figure 2.1 (Haase and Meyer, 1991): Pigment particles in the paint medium obstruct and scatter back incoming light, making a simple reflectance model inadequate.

Kubelka-Munk theory (KM theory) describes the scattering of light from pigmented materials. It provides equations to calculate the overall reflected colour of a particular pigment for which two reflectance coefficients, ***K*** and ***S***, are known. Haase and Meyer (1991) define the reflectance equation as:

$$R_\infty = 1 + \frac{K}{S} - \sqrt{\left(\frac{K}{S}\right)^2 + 2\frac{K}{S}}\ .$$

This result is a spectral curve in CIE XYZ colour space, which can then be converted to RGB for display on a monitor (Haase and Meyer, 1991).

It is not terribly intuitive for an artist to have to specify colours using ***K*** and ***S*** values. A preferential method would be to have the artist specify a desired colour and have the computer calculate the appropriate coefficients for them. Haase and Meyer (1991) achieve this using a standard colour picker and an iterative least-squares method to select the closest matching ***K*** and ***S*** values. Curtis et al (1997) take an alternative and more complex approach, asking the user of their application to define what their colour should look like over white and black swatches. This is fine if one has the desired pigments handy for reference, but again is unintuitive for the computer artist.

KM theory provides an accurate model for simulating pigmented materials such as paint. Indeed, Curtis et al report accurate results in their painting system. However, for the specific problem of watercolour, KM calculations may be wasteful. Haase and Meyer suggest that watercolour may be accurately simulated using the simple subtractive CMY model, but this introduces its own problems. I address the limitations of this colour model and formulate my own colour model for watercolour painting in section 7.2.

3. Fluid Dynamics

I present here a very brief overview of a mathematical model of fluid motion, specifically the Navier-Stokes equations for incompressible flow in order to familiarise the reader with the problem.

The motion of a fluid depends on the interaction between microscopic fluid particles. These discrete events give rise to apparently

continuous macroscopic behaviour. To put it another way, the collisions between the microscopic fluid particles look like smooth, rolling fluid motion when viewed at a large enough scale. It is this assumption that allows the derivation of a mathematical model for describing fluid motion.

The Navier-Stokes equations for incompressible flow look like this (Marsden and Chorin, 1979):

$$\frac{D\mathbf{u}}{Dt} = -\operatorname{grad} p' + \nu\Delta\mathbf{u}$$
$$\operatorname{div}\mathbf{u} = 0$$

Where p is the pressure of the fluid, $\rho = \rho_0$ is the mass of the fluid and is constant (the principle of conservation of mass is necessary to derive these equations), and **u** is the velocity of the fluid.

Stam (1999) presents an alternative, compact vector notation for the Navier-Stokes equations (only slightly more comprehensibly) as:

$$\begin{aligned} \nabla \cdot \mathbf{u} &= 0 \\ \frac{\partial \mathbf{u}}{\partial t} &= -(\mathbf{u}\cdot\nabla)\mathbf{u} - \frac{1}{\rho}\nabla p + \nu\nabla^2\mathbf{u} + \mathbf{f}, \end{aligned}$$

Where **u** is the velocity of the fluid, ρ the density, v the coefficient of kinematic viscosity and **f** is an external force that in most cases will be zero. Both forms of the equations also enforce the boundary condition, **u**=0 at the edges of the simulation volume. What this means is that the simulation must be entirely enclosed and fluid may not "leak out". Alternatively, the simulation may be wrapped around so that fluid leaving one side of the bounding volume re-appears on the other.

What these equations each say is that provided one knows the initial values for the velocity and the pressure, the state of the fluid can be evolved linearly over time (Stam, 1999), to put it another way, the motion of a given fluid can be calculated by stepping forwards through time and applying the Navier-Stokes equations each time step. The "look" of the fluid depends entirely on its viscosity. For a watercolour simulation, the density value can be interpreted as the amount of pigment in the fluid at that point.

The main problems in simulating fluid motion on a computer arise because the equations of motion must be chopped up into discrete steps of space and time in order to be simulated. Too large a time step and the simulation may "blow up"; too large a space step and the simulation will not yield accurate results. For engineering purposes it is necessary that extremely accurate results be obtained. For computer visualisation applications however, accuracy can be sacrificed for speed and stability as long as the results look like fluid motion to the casual observer (Stam, 1999).

In his 1999 paper, "Stable Fluids", Jos Stam presents just such a system [see accompanying CD-ROM for Andrew Nealen's 2D FFT implementation of Stam's method]. The fluid volume is described by a grid of fluid cells, or "voxels". Each voxel contains a known density and velocity value. Each time step, the solver updates the density and velocity values of each voxel in accordance with the Navier-Stokes equations.

The general Navier-Stokes equations are reduced to separate numerical methods covering each of the effects of viscosity, convection and velocity within the fluid to give a fast result. Stam ensures stability within the solver by back-tracing, essentially asking the question each time step, "Which of these possible results can be mapped back onto my initial data". The result is a fast, stable solver capable of producing very realistic, fluid-like motions.

4. Cellular Automata

Cellular automata (CA) are simple mathematical idealisations of complex systems. They consist of a lattice of discrete, identical sites (cells) holding arbitrary values. These values develop over discrete time steps, governed by rules determining each cell's interaction with its neighbours. (Wolfram, 1983). The idea is that given simple, microscopic rules, the cellular automata will produce complex, macroscopic behaviour over time.

A good basic example of cellular automata at work is Steven Conway's game, "Life". Conway uses a cellular automaton to produce behaviours analogous to the growth of bacteria.

In this model, each cell can either be alive or dead, represented as a binary 0 or 1. Each time-step, a cell's state is modified depending on the state of its neighbours. Gardner (1970) lists the rules as follows:

1. Survivals. Every cell with two or three neighbouring living cells survives for the next generation.
2. Deaths. Each cell with four or more living neighbours dies from overpopulation. Every cell with one living neighbour or none dies from isolation.
3. Births. Each empty cell adjacent to exactly three neighbours—no more, no fewer—is a birth cell. It becomes alive on the next time step.

Given these three rules, Life produces complex, self-organizing, repeating and even self-replicating "organisms" [see the accompanying CD-ROM for my own C++ implementation of Life), as well as a guide to some of the more interesting patterns].

4.2 A Cellular Automaton Model For Watercolour

In his 1990 paper, "Modelling Watercolor by Simulating Diffusion, Pigment and Paper Fibers", David Small approaches the problem by approximating only those properties of fluid motion that are important for the look of watercolour paint, and ignoring the rest. He assumes that simulating the rolling, turbulent motion of the fluid is not necessary for painting applications.

Small's set-up consists of a two-layered, rectangular cellular automaton. Each cell holds values denoting the amount of paint present in that cell, represented as a quantity of fluid and a quantity of pigment. Each cell also holds values for the amount of pigment and fluid that has been absorbed into the paper at that cell. The motion of paint on top of the paper, "Surface effects", and the motion of the paint within the paper "Substrate effects" are dealt with separately, and paint can move from the surface layer into the substrate layer.

The user adds fluid and pigment interactively at the start of the simulation and then watches the results evolve over time.

The movement of paint in the surface layer is defined by a displacement force, **D**, which is calculated for each cell in both the horizontal and vertical directions and consists of the forces of surface tension and "spreading" (diffusion). Essentially this means that water in a particular cell is pulled toward water over a 10-cell region, and also tries to balance the level of water in its direct neighbours. In the surface layer pigment moves along with the water in equal proportion to it.

Each time step, a new surface fluid value is calculated for each cell by first subtracting from the last step's value the fluid moving out of the cell due to the displacement force, **D**, and then adding on the fluid moving into the cell from its neighbours due to their displacement forces. Small constructs similar equations for the fluid and pigment moving in the substrate layer, and the absorption of pigment from the surface layer into the substrate layer; he also takes account of evaporation by simply removing a small quantity of fluid from each cell each step.

I implemented the techniques outlined by Small in a simple watercolour-painting program, with a view to improving the system to handle more effects. The simple image below [Fig. 3.2] shows an example of what can be produced.

Figure 3.2 *An image produced using the cellular automaton watercolour system. While mimicking some of the features of watercolour painting, the system suffers from visual artefacts, as well as being hard to control.*

The cellular automaton technique can accurately describe wet-in-wet painting. Unfortunately it gives undesirable visual artefacts (the grids of darker spots seen in the image above). These are caused by the fact that paint only ever moves horizontally or vertically in the simulation, and so the paint tends to group in dark grid patterns due to surface spreading. It is possible to reduce the strength of these artefacts by adjusting the surface tension and spreading parameters, but this causes the simulation to behave in unexpected ways.

Small's method is also limited to simulating wet-in-wet painting. The paint spreads over all cells in the simulation, as if one were adding paint to an already-wet surface. This makes it unsuitable for wet-on-dry painting and dry brush techniques.

However the biggest limitation of Small's method is its speed. In order to get it to run interactively, the simulation's cellular automaton had to be reduced in size to 60 cells square. In a 600 pixel square window, this means each cell is represented onscreen by 100 pixels. To produce detailed and believable watercolour painting, one pixel would need to be mapped to one cell at most, and preferably four to allow multi-sampled anti-aliasing. Rewriting to take advantage of multi-processor architectures could increase the processing speed of the automaton, but this is a special case and probably would not give the necessary speed boost anyway.

There are several ways in which the look of the simulation could easily be improved. For instance, a height field could be used to simulate the motion of the paint over the texture of the paper, simply by adding in an extra force to the displacement calculation. The grid-like visual artefacts apparent in the picture above could be removed by using a hexagonal cellular automaton similar to that described by Wolfram in his 1986 paper "Cellular Automaton Fluids: Basic Theory". However, this would further complicate the simulation, adding extra calculations for each time step, and for mapping the hexagonal grid back onto a rectangular display.

I now look at another technique, based on Small's, which includes a more physically-accurate fluid model and pigment model to produce a better visual result.

4.3 A More Accurate Model

In their 1997 paper, "Computer Generated Watercolour", Curtis et al describe another cellular–automaton-based model building on Small's approximation to include more realistic fluid and pigment simulation. As in Small's model, Curtis et al's model uses three layers of simulation, which they define as the shallow-water layer, the pigment-deposition layer, and the capillary layer [Fig 3.3, below].

An important addition Curtis et al make to Small's model is that of a height field to simulate the texture of the paper, which is generated by standard pseudo-random noise functions. The height field is used to define a gradient field that affects the motion of the shallow-water layer.

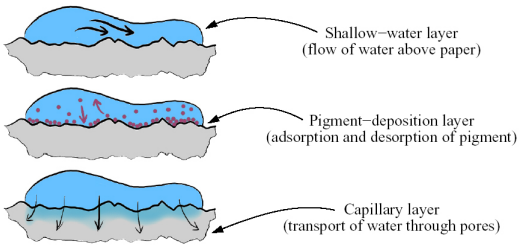

Figure 3.3 *(Curtis et al, 1997) Curtis breaks the watercolour simulation down into three discrete "layers" or processes.*

The shallow-water layer is an extension of Small's surface layer and is controlled by a fluid dynamics system to move the water, and hence the pigment suspended in it, around the paper. As described in section 2, a fluid simulation must enforce appropriate boundary conditions: in this case no fluid must move across the boundary of the simulation. Curtis et al achieve this by using a wet-area mask. This is simply a field of Boolean values defining whether or not the paper is wet at that point and is itself defined by the user's brushstrokes. Curtis et al simply set the velocity of water in any cell not inside the wet-area mask to zero. In a similar way to the method suggested by Small, Curtis et al move the pigment in the shallow water layer across cell boundaries in proportion to the water flow. Edge darkening is achieved by reducing water pressure at the boundary of the wet-area mask, causing water and pigment from the inside of the wet area to flow to the outside, thus producing darker edges.

The pigment-deposition layer controls pigment adsorption and "desorption": the movement of pigment from the water into the paper, and the movement of pigment from the paper back into the paint suspension. Curtis et al again use a more complex simulation of this process to achieve more believable results modelling the "staining power" and "density" of each pigment as well as using the height of the paper to scale the amount of pigment transferred (Curtis et al, 1997).

The capillary layer controls the movement of water within the paper itself and is analogous to Small's substrate layer. In Curtis et al's model it is only used to simulate back runs. Curtis et al use a very similar model to Small's for this layer.

5. Tackling The Problem Of Real-Time Interaction

As has been stated in the previous section, the biggest limitation of both Small's and Curtis et al's methods are the speed of simulation. While Curtis et al achieve very realistic and aesthetically pleasing images, his simulation—running at seven hours for a 640x480 image—is far too slow for an interactive application. Curtis et al allow the user to place the colour and water on the paper first, before running the simulation to produce the final image, but this seems unintuitive: artists rely on seeing the properties of watercolour evolve before their eyes as they place brushstrokes.

Clearly a strict physical simulation cannot produce acceptable results in real-time at today's processor speeds. It would seem that the biggest single speed limitation is caused by the fact that each cell of the cellular automaton must be visited each time step. This is especially apparent in my implementation based on Small's method. When the size of the CA is increased to say 600 cells square the frame rate plummets to about 10fps, just for drawing the canvas with no simulation. This is because the program must visit each of the 360,000 cells, convert their CMY-colour values to RGB and then draw a square at that point. Forcing each cell to be one pixel in size and then copying the colour values into a pixel array ready to be drawn to the frame buffer could gain a small speed improvement. It is important to realise however that it is the simple act of visiting each cell of the CA and performing some calculation there—no matter how simple—that is the limiting factor.

Therefore the most obvious way to improve the speed of the application would be to not visit every cell. One way to do this would be to keep a list of "dirty" cells and then each time step only those cells that need to be changed could be visited. However, in a painting application the whole canvas would quickly become "dirty" and so the speed improvement gained is quickly lost after a few brushstrokes.

It therefore becomes apparent that a completely different methodology is needed, the most obvious being a particle system. Particle systems have been successfully used to model a huge variety of natural phenomena such as fire, smoke and water.

What is important to the problem of watercolour simulation is that particle systems can easily be made to follow arbitrary vector fields, such as the gradient field describing a paper texture; they are easily understandable models of dynamic phenomena; and they are very fast to calculate given some common-sense optimisations.

I now describe my own proposed model for watercolour simulation using particle systems.

5.1 Modelling Watercolour Painting Using Particles

Particle systems cannot directly model fluid motion. Fluid dynamics calculations depend on the assumption that flow is continuous (Marsden and Chorin, 1979) and particles are, by their very nature, discrete.

In this case however, it is not necessary to have even a moderately accurate simulation as long as the result looks pleasing to the casual observer. In fact it is perfectly adequate to have the particles' motion defined by a static vector field representing the texture of the paper, as this is the most important effect on the fluid's movement.

In the proposed model the paper texture is represented by an array of vectors, $\boldsymbol{T}$, laid over the top of the image buffer, I. The vector, $\boldsymbol{T}_{x,y}$, for each pixel, $I_{x,y}$, is calculated from a height map, H, representing the paper's surface. In this notation, $H_{x,y}$ denotes the value of the height map in the x^{th} column and the y^{th} row of the array.

The vector for each pixel is calculated by taking the average of four unit vectors pointing towards adjacent pixels, each weighted by the difference in height between the neighbouring pixel and the central pixel, thus:

$$\boldsymbol{T}_{xy} = ([0\ 1].U + [1\ 0].R + [0\ -1].D + [-1\ 0].L\ ) / 4$$

Where:

$U = H_{x,y} - H_{x,y+1}$,

$R = H_{x,y} - H_{x+1,y}$,

$D = H_{x,y} - H_{x,y-1}$,

$L = H_{x,y} - H_{x-1,y}$

Hence $\boldsymbol{T}$ defines a gradient field over the paper's surface. The gradient field is averaged in order to simplify the calculations.

Paint is modelled as a particle system where each value has a colour, $\boldsymbol{c}$, and a wetness value, w. The wetness value is a model of the fluid carried by that particle and is used to define that particle's interaction with the paper. The paper itself also has an array of wetness values, W, corresponding to each pixel.

Particles are added to the simulation by the user using a virtual brush. Particles are born with a wetness value and colour value specified by the user. The particles are also given a starting velocity, $\boldsymbol{v}$, which carries them away

from the brush. Once entered into the simulation, particles are moved across the paper's surface by using the texture vector corresponding to the pixel in which the particle is currently residing ($\boldsymbol{T}_{x,y}$) as an accelerating force. The particle is then accelerated by a gravity vector, $\boldsymbol{g}$, so that paint dripping can be simulated, and a constant frictional term, f, is subtracted so that the particles tend to slow down over time. The velocity of the particles is then scaled by the wetness of the paper at that point and the particle's current position, $\boldsymbol{p}$, updated by adding the resulting velocity scaled by the time step, d_t, to ensure consistent results. The following pseudo code illustrates this process:

proc *updateParticleVelocities*(d_t)
  for all particles (i) **do**
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \boldsymbol{T}_{x,y}$
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i + \boldsymbol{g}$
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i - f$
    $\boldsymbol{v}_i \leftarrow \boldsymbol{v}_i * \boldsymbol{W}_{x,y}$
    $\boldsymbol{p}_i \leftarrow \boldsymbol{p}_i + \boldsymbol{v}_i * d_t$
  end for
end proc

The particles colour the paper as they move over it by taking a proportion of their colour, scaled by their wetness, and adding it to the colour value for the paper at that point. This simulates absorption of the paint by the paper. They also pick up a portion of the colour from the paper to simulate pigment being reabsorbed by the paint fluid.

At the end of each time step the paper and the paint are dried out to simulate evaporation by subtracting a constant term from their respective wetness values.

Wetness is added to the paper over the area of the user's brushstroke. By scaling the particles' velocity by the wetness value of the paper, effects such as edge-darkening and flow effects can be produced as the particles tend to move faster in the wet areas and bunch up in the dry areas at the edges of a stroke [Fig. 5.1, 5.2, below]. This is similar to Curtis et al's wet area mask.

Granulation also emerges from this system as particles are directed towards the low areas of the papers surface by the texture field, just as in real watercolour. My initial C implementation of this method can be found on the accompanying CD.

The most obvious shortcoming of this system in its present state is the graininess of the image produced. This is caused by the small size of the particles and their only affecting one pixel at a time. It would be relatively simple to "smudge" the particles' effect over an area larger than one pixel, just by averaging the particle's pigment deposition over the immediate neighbourhood of the pixel each time step.

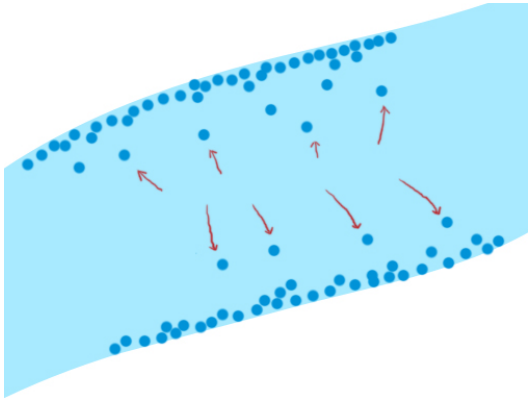

Figure 5.1: *Particles migrate from the centre towards the outside of a stroke where they are stopped by the edges of the wet area, causing edge-darkening.*

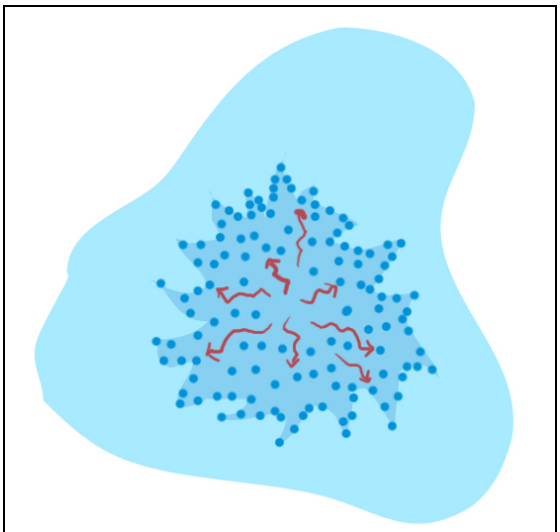

Figure 5.2*: In wet areas particles move freely following the texture field of the paper, creating flow effects*

However there is a more deep-seated problem with the particle-based approach: it is trying to model a continuous effect with small, discrete particles. The method described tries to get around this by spreading each particle's effect over time and space, analogously to paint depositing a certain amount of pigment on the paper as it moves over it. The fact is that this is just too far removed from the way watercolour works. In real watercolour painting it is more

important where the paint ends up rather than where it has been.

Additionally, using particles makes interaction between paint strokes very difficult. While I have attempted to simulate this by having the particles pick up colour from the paper as they move along, the results are poor. What is needed is some way to have the particles interact with each other. Whether using local fields, or trying to model collisions, the result of either would once again be slow computation and hence loss of interactivity.

A possible solution would be to use the particle system as a form of "marker" to build an implicit surface. This could be executed by interpolating the colour values between particles over the paper, or perhaps by implementing a 2D metaball system. Both of these methods throw up their own algorithmic problems and both would severely limit the speed at which the application would run.

6. An Assessment And Comparison Of The Methods Described So Far

So far I have investigated the methods described by Small (1991) and Curtis et al (1997) and I have also described my own particle-based method for simulating watercolour painting. I have implemented a simple painting program based on Small's cellular automaton technique as well as a quick test program for my particle-based method.

The measure of success for all of these methods has to be how well they replicate the watercolour effects described in section 2.1. For a watercolour-painting program it is also of utmost importance that the simulation runs in real-time.

So far the results have not been good. The cellular-automaton-based method excelled in simulating wet-in-wet painting and produced edge-darkening as an emergent result. I believe it could also make a fair approximation of all the other watercolour effects such as granulation and dry-brush with the addition of a height-field-based paper texture similar to that described in section 5.1.

It became apparent that the cellular-automaton-based method could not give real-time interaction at high resolutions (the program only runs reasonably well on a sixty-cell-square automaton). I considered this to be such a severe limitation that it was not worth continuing with the CA method and I should investigate other avenues. Curtis et al (1997) address the even more severe speed limitations in their system by having the painting and the simulation as two separate processes: first the user lays down areas of colour and water, and then runs the simulation for a set number of time steps to produce the finished image. The simulation time reported by Curtis et al for a 640x480 image is seven hours. For automatically painting a video frame (one of the applications implemented by Curtis et al) this may be acceptable if one has patience, but is hardly interactive.

In an attempt to produce an interactive painting program I designed the particle-based system described in section 5. While I have not had time to implement most of the ideas I presented, initial results have not been promising. As stated in the previous section, I feel that the possible improvements mooted would all incur a severe speed hit, for questionable improvement in visual quality.

Essentially the problem as I have found it comes down to this: one can have a realistic and visually interesting simulation that is very slow, or one can have an interactive application that does not look very much like watercolour! I believe I failed because I attempted to resolve an intractable problem. In order to get a realistic simulation of watercolour, one has to be prepared to wait for the simulation to execute.

With that in mind I now present one last possible method for simulating watercolour painting.

7. Watercolour Fudge

Based on my assessment of existing methods and my own implementation in the last section, I have decided that simulating watercolour effectively and interactively on today's hardware is not a realistic proposal. I therefore propose a method to fake it. While this method does not attempt to simulate the process of watercolour painting as an experienced watercolorist would expect, it provides a framework for effectively reproducing all the basic effects of watercolour painting (glazing, wet-in-wet, dry-brush etc) in a controllable and, most importantly, interactive manner.

7.1 Blocks of Colour

The proposed framework is based on the observation that watercolorists work by blocking in areas of colour (Paramón, 1993; Bolton, 2000). Separate areas of colour may merge and become one as the painting progresses, but when

they dry, these areas dry individually and so may be treated as separate units.

My idea is to have a program where the user lays down areas of colour (*blocks*) with standard brush tools. The user then decides how each block will interact with blocks below it by choosing a *blending mode*. This decision is analogous to the real watercolour artist deciding how long to leave an area of paint to dry before painting over it. Here, of course, the virtual watercolorist is gifted not only with the power of an undo function, but by having the program keep the original blocks in memory, the user may re-order blocks and change the blending modes at will. This working paradigm also sits better with the natural painting style of many artists, who would perceive a painting as being made up of separate areas of colour representing say a house or a tree. To an observer of the finished painting it is the interaction of the areas of colour that is more important than the interaction of pigment grains at the microscopic level.

Having decided that the user will work with flat areas of colour, it is now important to look at how the program captures these *blocks* and makes them look like watercolour.

As the user draws strokes on the paper, the painted area could be captured either using a bitmap mask to define the pixels touched by the brush, or using vectors to define the area filled in. Bitmaps are advantageous because they lend themselves easily to standard image processing techniques, whereas using vectors would eliminate aliasing problems and would be resolution-independent.

Once captured, the block can be stored in memory in a similar fashion to a layer in Photoshop. Then the user can decide in what manner each block should be blended with the blocks below it. This problem should be approached by taking each of the basic effects and techniques described in section 2.1 and formulating a method to reproduce these effects using standard image-processing techniques:

A) Wash

A wash could be effected simply by having the user place multiple strokes of varying colour to describe the wash (flat, graded, variegated) and then applying a Gaussian blur filter with a large kernel to blend the strokes together into one block of smooth colour

B) Wet-into-wet and back runs

Here the upper block must bleed smoothly into the lower block. A simple way to do this would just be to use a Gaussian blur again to blend the two blocks together. A more accurate method would be to use a particle system similar to that described in section 5.1.

Here however the particles should be placed along the boundary of the top block (this is trivial to accomplish with a vector-based block: one could simply re-sample the shape outline to any desired accuracy) before being released into a suitable vector field such as that described in section 5.1, or perhaps one generated by a Perlin-noise-variant. The boundary of the block moves with the particles.

The block must lighten in colour to reflect the fact that its pigment content is now spread over a larger area. This can be accomplished simply by scaling the colour of the entire block down in proportion to the difference between its old and new area. To finish the effect a Gaussian blur could again be applied to smooth the boundaries between the upper and lower blocks.

C) Dry brush

Dry brush would have to be implemented at the time the user made the stroke. This effect would be very simple to implement: as the user draws a brushstroke, colour is only applied to the raised parts of the paper.

D) Edge darkening

Edge darkening can be achieved by contracting the block slightly, darkening the area that is the difference between the two and slightly lightening the area in the middle.

Once these effects have been applied all the blocks can be combined together to make the final flat image in the frame buffer. Now the last stage should be applied:

E) Paper texture and granulation

Paper texture can be applied to all blocks simultaneously by multiplying the frame buffer pixel colour value by the value of a paper texture height field. Thus it would appear that raised areas had received less pigment, and pigment had settled into lower areas. Granulation is simply a repeat of this process using a paper texture that has been clamped in order that only the lowest areas of the paper are darkened. The strength of both these effects should be dependent on the colour of the pixel being multiplied, to simulate the effects caused by different pigments having different weights.

Here a realistic wash has been created. First a series of graded brush strokes were smoothed with Gaussian blur, and then the image was multiplied with a paper texture image.

Here a single stroke has been treated with two different effects: dry brush and edge darkening. Each effect was masked by blurring the stroke and then using the difference between the blurred stroke and the original as the mask. The edge darkening effect was created by duplicating the stroke, masking the duplicate with a thin mask and then multiplying it back over the original. The dry-brush effect was created by clamping the paper texture to produce large blotches and then using these blotches to cut out the bottom part of the stroke. Again a paper texture has been applied to the whole image, but in this case a second paper layer has been multiplied over. This second layer was clamped and blurred to create small dark spots simulating granulation in the low areas of the paper's surface.

While there has not been time to implement this method, the two images above were produced as proof of concept. They were produced by using Adobe Photoshop to create the areas of colour, then applying the same image processing techniques described above to produce the watercolour effects.

7.2 The Colour Problem

In real watercolour painting, laying the same pigment over the top of itself will produce a colour only as dark as the dry pigment, which in many cases will still be a strong, vibrant colour.

However in subtractive colour mixing on the computer, the same colour laid onto the same pixel several times can quickly tend to pure black.

The problem arises because in the computer's colour representation the user is limited to a palette of three colours determined by the RGB display palette, or its linear transformation: CMY.

Curtis et al (1997) solve this problem by having the user define pigments in terms of the Kubelka-Munk coefficients ***K*** and ***S***. The pigments are then applied to the paper in individual glazes.

A more intuitive solution would be to give the user a virtual paint box, resplendent with a number of basic pigments. The user can then mix these pigments on a virtual palette to produce the desired colour before applying it to the paper. In order to do this it would be necessary to subvert the standard RGB display model and have a number of colour planes that correspond to the choice of pigment. Obviously, the number of pigments available to the user would be limited by the amount of available memory. Defining a colour in terms of, say nine different pigments would require three times as much memory as in the RGB model. This should not be too much of a problem, as one can safely assume a graphics user to have at least 512MB of RAM. In cases where RAM is a limiting factor, it should be sufficient to write a simple paging system to swap blocks that are not in use to disk to free up memory.

Each pigment should initially be defined in the CMY model, as then it is a simple calculation to combine them back into RGB space for display on a monitor, or for saving in a standard image file format.

The main advantage of this colour system is that it directly corresponds to the way in which an artist mixes colours using real paints. Unlike the Kubelka-Munk model, it gives users an easily-understandable way to mix colours for painting.

8. Conclusions

I have researched and described the implementation of three different methods for simulating watercolour painting. My research has led me to investigate two existing systems for simulating watercolour painting, namely those of Small and Curtis et al as described in their 1991 and 1997 papers, respectively. I have covered, in detail, the related topics of fluid dynamics, cellular automata and Kubelka-Munk theory, as well as the history and practice of the medium itself.

The first implementation I made was based on the method described by David Small (1991) using a cellular automaton to model diffusion of paint over a flat surface. I had originally planned to address the limitations of Small's system and extend it to incorporate a model of the paper's texture for a better visual result. The initial visual results were fairly good, despite some artefacts. The system realistically portrays painting with a wet medium, although it feels more like pushing paint around in a palette than painting on paper.

Despite this initial promise, it was apparent that using a cellular automaton was far too slow to create an interactive painting program. The simulation has to run at a very low resolution to give real-time interaction. It is possible that with some more time to look into this it might be possible to optimize the cellular automaton calculations. However this optimization would need to result in a speed increase of nearly one hundred times to yield real-time interaction at high resolutions. I considered this to be too tall an order and decided to look into other methods.

Based on my experience with the cellular automaton method, and the research I had done into fluid dynamics, I then tried to approximate fluid motion using particles. Conversely to my experience with the cellular automaton, I now found that I had real-time interaction, but the visual result was very poor. I hypothesized some possible solutions, but none seemed likely to give the visual result I was looking for.

It seemed with these methods of trying to approximate the actual dynamics of watercolour I was faced with an either-or situation: I could either have a pleasing, accurate visual approximation of watercolour, or I could have real-time interaction. Not both.

Determined not to be defeated, I took one final look at the problem from a different angle. Since trying to simulate or approximate the physical properties of watercolour had been unsuccessful, I formulated a new method based entirely on my own research using watercolour paints [see accompanying CD2] and my research into techniques applied by the traditional watercolorist (Parramón, 1993; Bolton, 2000).

The result is an innovative, practical system. It employs standard image processing techniques that are easy to implement and fast to execute. It also incorporates a proposal for a new way of representing colour that directly corresponds to colour mixing and selection in the real world. While the proposed system does not give the user a straight simulation of watercolour painting per se, my motivation for this project has been to come up with a system that can produce believable and aesthetically pleasing results. If I had time to implement this system, I believe it would do exactly that, and I believe that the test images on the previous pages demonstrate that the concept would work extremely well when implemented.

References

GARDNER, M., 1970, Mathematical Games: The fantastic combinations of John Conway's new solitaire game "life", *Scientific American*, 223 (October 1970), 120-123.

CURTIS, C. J., ANDERSON, S. E., SEIMS, J. E., FLEISCHER, K.W., AND SALESIN, D. H, 1997, Computer-Generated Watercolor, *Proceedings of Siggraph '97*, 421-430.

SMALL, D., 1991, Modelling Watercolor by Simulating Diffusion, Pigment and Paper Fibers, *Image Handling and Reproduction Systems Integration,* SPIE 1460 (1991), 140-146

WOLFRAM, S., 1983, Cellular Automata, *Los Alamos Science*, 9 (Fall 1983), 2-21

WOLFRAM, S., 1986, Cellular Automaton Fluids: Basic Theory, *Journal of Statistical Physics*, 45 (November 1986), 471-526.

STAM, J., 1999, Stable Fluids, *Siggraph '99 Conference Proceedings,* 121-128

STAM, J., 2003, Real-Time Fluid Dynamics For Games,

HAASE, C.S. AND MEYER, G.W., 1992, Modeling Pigmented Materials For Realistic Image Synthesis, *ACM Transactions On Graphics*, Volume 11, Issue 4, 305-335

ODDY, R.J. AND WILLIS, P.J., 1991, A Physically Based Colour Model, *Computer Graphics Forum*, Issue 10, 121-127

BOLTON, R., 2000, Creative Watercolour Techniques, London, Search Press

HARNSON, H., 2000, Watercolours In A Weekend, London, Search Press

MARSDEN, J.E. AND CHORIN, A.J., 1979, A Mathematical Introduction To Fluid Dynamics, 3rd Edition, New York, Springer-Verlag

PARRAMÓN, J.M., 1993, The Complete Book Of Watercolour, London, Phaidon Press

APPENDIX A – Contents of the accompanying CD-ROMs

CD1

bin

particleWatercolour.exe
- Test program for particle method described in section 5.1

caWatercolour.exe
- Implementation of cellular automaton method described in section 4

gameOfLife.exe
- Implementation of Conway's cellular automaton game, Life

Dance.exe
- Implementation of behavioural system from Appendix B

flowanim.exe
- Andrew Nealen's implementation of Jos Stam's stable fluid solver

docs

- electronic versions of this paper in MS Word and PDF format

source

- Accompanying source code and MS VC.NET project files for the software found in **bin**

CD2 – Video Reference

Contains four video files in DivX 5.02 codec (provided on CD1) showing my experimentation with watercolour paints, and trying to reproduce some of the effects described in section 2.1.

Dance! – A Nightclub "Battle" Simulation

Before starting work on the watercolour simulation, I spent some time looking into behavioural systems and artificial intelligence, particularly the concept of an "autonomous agent".

My idea was to set up a fictitious nightclub situation in which agents would compete against each other to see who had the best dance moves. How each agent faired would affect their behaviour within the nightclub and towards other agents.

I started out by looking into what are known as "steering behaviours": extensions of the flocking algorithm devised by Craig W. Reynolds. Reynolds and others have extended the range of behaviours to include collision avoidance, seek and pursuit behaviours, wandering and many more.

I had planned to implement the dancing agents as a sort of state machine. The agents' behaviour would then be controlled by a weighted combination of steering behaviours, for example **seek and arrive** at the bar to get a drink, **avoid** other dancers and walls. The active behaviours are kept on a list or vector which the state machine updates to effect the agent's objective.

I got as far as implementing all of the applicable steering behaviours, and the active list for executing them within the update() function of each dancer, or "boid", object. I achieved this using the inheritance and polymorphism features of C++. Each behaviour is derived from an abstract Behaviour base class. Each boid then keeps a list of Behaviour base class pointers and calls their update() function each time step to get a steering force due to that behaviour. The steering forces are combined with a weighted average to make an acceleration force with which to alter the boid's velocity

There was a small problem created due to the fact that some behaviours require a list of boids on which to operate. For example the collision avoidance behaviour needs to know where all the other boids are in order to calculate a steering force to avoid them. The collision avoidance behaviours were incorporated directly into the boid class, rather than being weighted like the rest of the behaviours, to ensure that boids would never intersect each other or go through walls.

Thus the collision-avoidance behaviour objects have to "know" about the boid objects and vice versa, creating a declaration problem. The solution was to inherit the Boid class from an abstract base class as well, thus allowing the behaviour objects to store a pointer to this base class and access data such as the boids' position and orientation, while still declaring the actual Boid class itself after the behaviour classes.

I also began implementation of the virtual DJ. The idea was that there would be a DJ class that would mix mp3 music files on-the-fly to simulate a real nightclub experience. The dancers would be animated to move in time to the music. I got as far as having the dancers flash in time to the music playing. Rather than trying to beat-match songs at run-time, the approach taken was to use music-editing software to set the BPM of each song beforehand, then the problem of mixing the songs together was reduced to keeping track of how many beats had passed using a timer, and changing the volume of each song appropriately. Animating the dancers in time to the music can be accomplished easily since if the program knows the BPM of each song it is trivial to time the animation to match.

I think I made a very good start on this project. Although there is no implementation of the state machine for each agent, the steering behaviours and animation timing work well. I decided to change project and investigate watercolour primarily because I found that my main interest in the dance simulation was in the music and the virtual DJ. While this is a fairly challenging area to develop, it seemed to me to be a little too far removed from graphics and animation, and I wanted to tackle a project that aligned more directly with my current interests in computer graphics.