DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING, SECOND CYCLE, 30 CREDITS

STOCKHOLM, SWEDEN 2016

Simulating High Detail Brush Painting on Mobile Devices

Using OpenGL, Data-Driven Modeling and GPU Computation

ADRIAN BLANCO

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF COMPUTER SCIENCE AND COMMUNICATION

Simulating High Detail Brush Painting on Mobile Devices

Using OpenGL, Data-Driven Modeling and GPU Computation

ADRIAN BLANCO

Master's Thesis in Computer Science (30 Credits)
Supervisor: Mario Romero
Examiner: Tino Weinkauf

Abstract

This report presents FastBrush, an advanced implementation for real time brush simulation, which achieves high detail with a large amount of bristles, and is lightweight enough to be implemented for mobile devices. The final result of this system has far higher detail than available consumer painting applications. Paintbrushes have up to a thousand bristles. Adobe Photoshop is only able to simulate up to a hundred bristles in real-time, while FastBrush is able to capture the full detail of a brush with up to a thousand bristles in real-time on mobile devices. Simple multidimensional data driven modeling is used to create a deformation table, which enables calculating the physics of the brush deformations in near constant time for the entire brush, and thus the physics calculation overhead of a large number of bristles becomes negligible. The results show that there is a large potential for use of data driven models in high detail brush simulations.

Referat

Simulering av penselmålning med hög detaljrikedom på mobila enheter

Denna rapport presenterar FastBrush, en avancerad implementation för realtidssimulation av penselmålning som uppnår hög detalj med en stor mängd penselstrån, samt är snabb nog att implementeras för mobila enheter. Det slutgiltliga resultatet av denna implementation har mycket högre detail än nuvarande tillgängliga konsumentapplikationer. Penslar har ett tusen individuella penselstrån. Adobe Photoshop är begränsad till att simulera maximum ett hundra penselstrån, medan FastBrush kan uppnå fullständig detaljrik återgivning med upp till ett tusen penselstrån i realtid på mobila enheter. Enkel multidimensionell datadriven modellering används för att skapa en deformationstabell, vilket möjliggör att beräkna fysiken för penselns deformation i nära konstant tid, och därför blir de kostnaden av fysikkalkylationerna för ett högt antal individuella penselstrån försummbar. Resultaten visar att det finns stor potential för användning av datadrivna modeller i högdetaljerade penselsimulationer.

Acknowledgements

I would like to thank Jeasmine Ljungström and Huiting Wang for contributing with the paintings showing the capabilities of this project, and Mario Romero for his never ending support and encouragement to achieve the impossible. I would also like to thank everyone at Bontouch who have made this thesis possible.

Contents

0.1 Glossary

- **Natural media:** Physical objects used for art such as a brush or a canvas. In contrast to digital media.
- **Brush detail:** Brush detail (and equivalent terms) in context of a brush simulation will refer to the amount of bristles of the simulated brush, unless otherwise stated.
- **Brush:** A whole paintbrush, but will often be referred to as the bristle part of the brush unless otherwise stated.
- **Handle:** The wooden handle of the paintbrush, most often referred to as the part of the handle which connects directly to the bristles unless otherwise stated.
- **Bristle:** An individual hair on the brush.
- **Pixel:** The smallest possible addressable element on a screen.
- **Frame:** A fully rendered and composited image, which the user sees. Many frames in a row gives the appearance of motion, such as in a video.
- **Frame Buffer:** Storage for a partly or fully composited image.
- **FPS:** Frames per second, often the maximum update rate of the system.
- **Key-value:** a key-value is a surjective mapping, which for each key has a corresponding value.
- **6DOF:** Six degrees of freedom, meaning spatial expressiveness of position in the x, y and z axis with and angle of pitch, yaw and roll.

Chapter 1

Introduction

Ralph Mayer states in The Artist's Handbook of Materials and Techniques:

> "There is no item of greater importance to the successful execution of a painting than a sufficient quantity of the very-highest-grade brushes that it is possible to find. It is one department of the artist's equipment where no skimping or compromise should be allowed: one may go without or use makeshift supplies of some items but poor brushes are a severe handicap to good painting" [1].

Using proper brushes is critical for a good painting, enabling competent artists to create without limitations. Digital artists can greatly benefit from having brushes with high expressiveness when painting using digital painting programs [15]. Commercial painting applications have generally been limited to very simple painting methods, such as repeating 2D textures along a path. While such systems can give the visual appearance of natural media, they do not behave like natural brushes. Brush simulation systems instead seek to simulate a virtual brush which behaves like a brush would in real life when painting with natural media, and use that as a basis for visual results. However, many of the state-of-the-art brush simulation systems do not take into account real-world constraints, and are not suited for mobile devices with very limited performance capabilities. As mobile devices have become an integral part of our lives and large part of our digital interactions are now on mobile devices, developing advanced consumer applications with heavy performance constraints in mind is important. An ideal virtual brush could be described as both lightweight in performance requirements, and high in detail. Howevever, as many state-of-the-art brush simulation systems require high computational resources to run, they are also often limited in the amount of detail a brush is able to express for real-time rendering [10].

1.1 Problem

A problem in brush simulation is that a lot of the algorithms, systems and simulations are not fit for use in industry or consumer applications, and many of them require specific hardware and would not perform well on devices with lower performance [10]. While there exists limited research of brush simulation methods suitable for lower performance devices, high detail brush simulation with a large number of bristles for lower performance devices is an area where little research has been conducted.

1.2 Objective

The goal with this report is to examine existing methods of brush simulation, choose a set of methods suitable for high detail brush simulation for mobile devices, and evaluate them in comparison with other methods.

1.3 Delimitations

When developing a fully realistic brush painting system, there are many different factors involved which have to be taken into account. If we focus on the three main components, the brush, the ink and the paper, there is a multitude of interactions between them. A set of possible interactions is shown in fig 1.1 and detailed below.

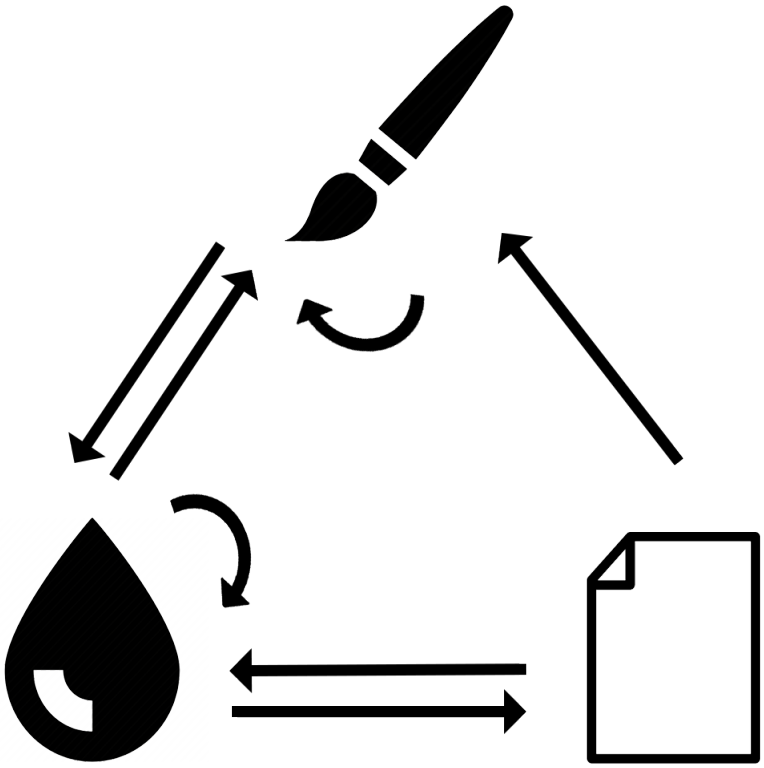

Figure 1.1. A set of possible interactions in brush painting

- **Brush to brush:** When simulating bristles in a brush, one of the most significant factors one must consider is the state of the other bristles. Bristles have mass, they take up space, and these properties must be considered in order to ensure that the bristle deformation takes into account collision with other bristles [10][15].
- **Brush to ink:** When simulating the brush, moving the brush should also move around both the ink on the brush and also the ink on the canvas on a physical level [16].
- **Ink to brush:** Ink has certain properties which can affect the physics of the brush, for example a wet brush has the properties that bristles can stick together changing the physical movement of the bristles [15].

- **Ink to ink:** When ink comes into contact with ink the simulation should simulate the way the fluids mix and spread [16].
- **Ink to canvas:** The ink should be imprinted onto the canvas [12].
- **Canvas to ink:** The properties of the canvas should determine how the ink behaves, such as the way it spreads or dries [16][18].
- **Canvas to brush:** The brush should deform according to it's contact with the canvas [12][9].

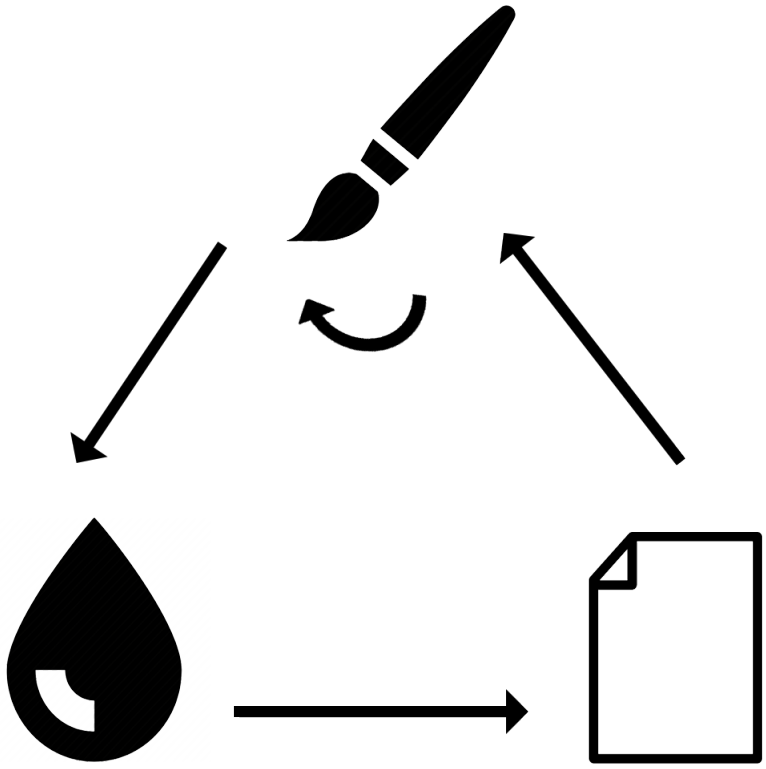

Figure 1.2. A limited set of possible interactions in brush painting

This does not include interactions such as the brush deforming the canvas, or other types of changes to the canvas such as canvas physics, as it is of little relevance to research on brush simulation. Implementing all of these interactions in the simulation would be far too time consuming to be included in the scope of this thesis, and thus I have chosen to focus on two of these interactions. These two are the brush to brush, and canvas to brush interactions, which are the most important interactions in order to accurately capture the deformation of the brush [15]. Furthermore, the

brush to ink and ink to canvas interactions need to be implemented at a minimal level in order for the user to be able to paint with the simulated brush.

Chapter 2

Background

2.1 Available technologies

There are many different methods to choose from when implementing brush simulations. A brush is not simply one component, but instead many different components working together, and different components behave in different ways with different results. Therefore, with the combination of possible technologies used by brush algorithms there is significant overlap between the technologies used in papers. The chart below from "A Brush Stroke Synthesis Toolbox" [12] displays a quick overview of state-of-the-art papers and their technologies categorized.

These are as follows: **Output**: raster (R) or vector (V). **Strategy:** acquisition of real brush footprints (A), physical simulation (S), data-driven methods (D), or procedural methods (P). **Spline:** for work that models splines, are they discrete (D) or continuous (C). **Brush:** for work that models a brush head, is it a mesh (M), skeleton (S), interpolated geometry (I), or individual bristles (B). **Solution:** systems can be solved via integration (I), energy minimization (M), or data-driven methods (D). **Tween:** short for in-between, intermediate states between simulation steps can be generated by stamping (T) or sweeping (W) [12].

2.1.1 Output Type

The output for the brush simulation algorithm is either raster based (R) or vector based (V). Raster based output is a bitmap or 2D grid of pixels, most solutions use raster based output. Vector based output uses a series of parametric curves instead of pixels to represent the brush strokes. Vector based output can often be more complex to implement and have limitations in the types of strokes they can represent, but their advantage is that they are resolution independent and require less data to be represented [12].

work	output	strategy	spline	brush	solution	tween
Adobe Illustrator	V	P	–	–	–	W
Adobe Photoshop	R	P	–	–	–	T
Ambient Design ArtRage	R	P	–	–	–	TW
Bai et al., 2007	R	P	–	–	–	T
Baxter at al., 2001	R	S	D	M	I	T
Baxter and Govindaraju, 2010	R	D	C	M	D	T
Baxter and Lin, 2004	R	S	D	MI	M	T
Chu and Tai, 2004	R	S	D	S	M	T
Chu, 2007	R	S	D	S	M	W
Corel Painter	R	P	–	–	–	TW
DiVerdi at al., 2010	RV	S	C	B	I	TW
Lu and Huang, 2007	R	S	C	B	M	T
Mi et al., 2002	R	P	–	–	–	W
Okabe et al., 2005	R	A	–	–	–	T
Pudet, 1994	V	P	–	–	–	W
Saito and Nakajima, 1999	R	S	C	M	M	T
Van Laerhoven and Van Reeth, 2007	R	S	D	MI	M	T
Vandoren et al., 2009	R	A	–	–	–	T
Vandoren et al., 2008	R	A	–	–	–	T
Xie et al., 2010	R	P	–	–	–	W
Xu et al., 2004	R	D	–	MB	–	T

Figure 2.1. A chart of the different brush simulation technologies

Source: A Brush Stroke Synthesis Toolbox, Image and Video-Based Artistic Stylisation [12]

2.1.2 Algorithm Strategy

The algorithm strategy is responsible for generating brush strokes, and is one of the defining traits of a brush simulation system. Procedural strategies (P) are fast and very simple methods to generate brush strokes without having to model a virtual brush. This strategy is used in most commercial painting applications. Simulation strategies (S) are instead based on calculating and approximating the dynamics and physics as close to a real brush as possible. Acquisition strategies (A), on the other hand skip expensive brush simulations and use custom hardware in order to measure the shape and properties of a real brush in real-time. Data driven simulations (D) are a mix of both, a database of shapshots of real brushes under different conditions are created and used as a basis for simulating arbitrary brush shapes [12].

2.1.3 Spline Model

For systems that use a spline representation of the brush, the spline model determines whether the splines are discrete (D) or continuous (C). Discrete splines lend themselves very well to computing physics or dynamics, however many discrete points might have to be used in order to achieve acceptable quality. Continuous spline models are less suited for physics calculations or dynamics, however the resulting spline is always smooth [12].

2.1.4 Brush Model

The brush model determines how the brush is represented inside the simulation with varying degrees of complexity. The most simple model is to represent the brush head as a bulk triangle mesh (M), which wraps around one control spline. A more advanced model could use multiple branching control splines (S) as a skeleton for meshes, in order to control the potentially asymmetric behaviour of a brush. If the brush head is represented as individual bristles instead of meshes, individual bristles (B) can be represented as splines for high fidelity at a great computational cost. A way to increase performance is to only simulate a few individual control bristles (I) and use them to interpolate the rest of the bristles [12].

2.1.5 Solution Method

When computing the solution to the physics of the brush dynamics, there are a number of methods available. The most straightforward way is to update the shape based on the integration (I) of internal and external forces over time. As brushes tend to return to their rest shape quickly, a common solution is to compute the dynamics using energy minimization models (M). Potentially fastest of all solution methods are data driven models (D), which can solve brush shapes without any costly calculations involved [12].

2.1.6 Tweening

Tweening refers to the methods used to extract the resulting imprints from the brushes. Stamping (T) is a numerical approach which imprints the brush many times along the stroke path onto the canvas, in order to create the appearance of continuous strokes. Sweeping (W) is an analytical approach which calculates the resulting imprint from the differences in the brush geometry between each frame [12].

As we can see, there are large variations of technologies from paper to paper. Furthermore, not all papers explicitly put focus on all steps, but rather they focus on presenting one method, which they have performed research on. In many cases, the papers are presenting an implementation and then letting the results speak for

themselves. Therefore, I will try to describe and group the papers in terms of the methods they have chosen. In many cases, the methods are not very different from another and thus the focus will be on the results.

2.2 History of Brush Simulation

Early methods: The earliest method of brush simulation and also one of the most commonly used today is to stamp 2D bitmaps along a brush path [15]. [Strassmann, 1986] presented a system where brush paths are generated based on user input, and then dynamically textured based on defined parameters to generate a brush strokes [5].

Brush simulations: The first fully physically-based 3D brush model can be said to be the calligraphy brush proposed by Saito in [Saito and Nakajima, 1999][15], it was based on a system of energy minimization with a single control bristle and sweeping circles around the control bristles as the brush volume [6]. [Chu and Tai, 2002] introduced improvements which would allow simulation of brush flattening and spreading due to incorporating lateral nodes in their brush simulation [14]. [Baxter and Lin, 2004] presented a system which allows simulation of a small amount of control bristles with simulated dynamics, with interpolation of a large amount of bristles with a system for computing dynamics for them [15].

A lot of brush simulation methods have not been suitable for industry use because of the limitations of the simulation or unacceptable performance or detail, [DiVerdi et al., 2010] presented a system for brush simulation which despite simulating each bristle individually is able to achieve high performance through improved simulation methods [10]

Acquisition based models: Greene [Greene, 1985] introduced a prism based system for acquiring the results of brush movements on a surface as one of the earliest examples of data driven brushes [7]. [Vandoren et al, 2009] further expanded on this concept with custom hardware as a canvas, and acquiring imprints from real brushes applied onto the canvas as basis for the simulated imprints [8].

Data driven models: [Xu et al., 2004] created brush simulations based on bristles "macros" where a macro would represent a bunch of bristles clumped together according to data gathered from offline simulations [11]. [Baxter and Govindaraju, 2010] presented a data driven system which recorded the characteristics of a brush offline and used them as a basis for quick simulation [9].

2.3 Related Work

Consumer painting applications have for a long time been limited to simple methods of repeated static 2D textures imprinted on the canvas in order to generate a stroke [2]. Consistently being able to draw smoothly and predictably with stable and robust behaviour is absolutely necessary for consumer applications, and the advantages of these widely used simple models is that they satisfy these constraints [10]. Research in brush simulations have produced impressive and highly advanced results, however those often require specific setups or high end hardware. Those results have not been adopted by consumer applications to date because of the strict requirements for professional use and have remained in the domain of research [10]. However, there are few papers which have researched brush simulation algorithms for consumer applications in the perspective of performance and stability required for implementation on a wide range of hardware with diverse performance.

"Simple Data-Driven Modeling of Brushes" by Baxter and Govindaraju was a result of Microsoft Research researching ways to implement data driven models for fast brush simulation in Microsofts painting Fresh Paint [9]. "Industrial-Strength Painting with a Virtual Bristle Brush" by DiVerdi et al [10], was a result of Adobe researching brush simulation technologies to implement a brush simulation system for Adobe Photoshop [10]. Both of these papers detailed the results of trying to develop state-of-the-art brush simulation systems to be used in consumer applications, and most importantly focusing some on the real-world limitations which might not be as important within research.

2.3.1 Simple Data-Driven Modeling of Brushes

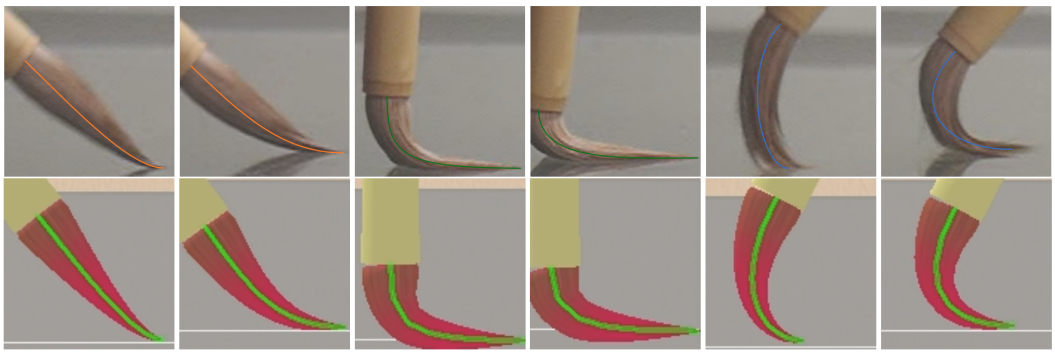

Figure 2.2. Sample of the real world brushes, and their respective data driven simulated counterparts

Source: Simple Data-Driven Modeling of Brushes [9]

[Baxter and Govindaraju, 2010] uses a small library of brush properties, which have been gained through deforming a physical brush based on parameters and then analyzed. Through a very small library size, expressive simulations are achieved

with only an interpolated constant lookup for the position of each bristle, which enables this system to run at orders of magnitude faster than other systems. Their method uses pictures of real life brushes during different deformations and handle parameters with two degrees of freedom (height from brush to canvas and angle from brush to canvas). From each picture a bezier curve is approximated through the brush which serves as the basis for the deformation table. The deformation table consists of the bezier curves representing the deformation for all of the pictures. A new model is generated by taking in handle parameters, and bilinearly interpolating between models which correspond to neighbouring handle parameters. This bezier curve is considered the static equilibrium of the brush. Dynamics are calculated to adapt the brush model to new configurations, as friction is calculated as a simplified cone shaped energy well following the Coulomb model of friction. Just like [Chu and Tai, 2002], brush tip spreading is explicitly calculated based on the compression ratio and the bend angle of the brush. Previous methods such as energy minimization methods used by [Saito and Nakajima, 1999] or [Baxter and Lin, 2004] have been very costly compared to data driven methods such as this. The brush is represented as a mesh, which wraps around the control bristle based on it's deformations [9].

2.3.2 Industrial-Strength Painting with a Virtual Bristle Brush

Figure 2.3. Sample of the real world brushes, and their respective data driven simulated counterparts

Source: Industrial-Strength Painting with a Virtual Bristle Brush [10]

[DiVerdi et al, 2010] explores the trade-off between accurately simulating individual bristles with real-time constraints. They implemented their brushes with strands as particles connected by constraints, which is effective in efficiently capturing bristle dynamics in straightforward ways. Bristle dynamics are computed using an efficient solver for sparse linear systems, which only takes one iteration for a solution unlike other iterative solvers which is the basis for the achieved real-time performance. Their system manages to simulate up to a hundred individual bristles, using a system which is able to perform well even with low performance hardware.

2.4 Detail in brush simulation systems

In mesh based brushes such as those used in [Baxter and Govindaraju, 2010] the entire brush is a single bulk surface, which is deformed by the physical deformation of the control bristles that the mesh wraps around. Meshes are fast to render on modern graphics hardware, however since the brushes are represented as a single mesh wrapped around a control bristle, they are very limited in their expressiveness. They depend on texture mapping in order to give the illusion of individual bristles, and they do not support split or distressed brush strokes. When using skeletal meshes, there exists major limitations in that the meshes are solid volumes and thus the imprint will only be of solid volumes instead of the containing bristles that they are simulating. There are ways to enhance the imprints so that they look more like individual bristles than solid volumes, such as using a texture wrapped over the skeletal mesh which attempts to simulate an imprint [20], however texture stamping on a skeletal mesh is very limited in expression as it's trying to emulate the imprint of a multiple individual bristles on one volume. Another solution is to split the skeletal mesh into smaller skeletal meshes which simulate bundles of bristles such as [Xu et al., 2004]. This solution is more expressive, as each skeletal mesh is deformed by its spine, a skeletal mesh is only as expressive as a single bristle in theory. By increasing the amount of skeletal meshes the expressiveness increases, however it is still far from the the detail achieved by simulating individual bristles. In [DiVerdi et al., 2010] a criteria for their implementation was to capture the fidelity and detail of real brushes, and thus representing their brush as individual brushes was necessary to achieve that goal.

Systems which rely on simulating individual bristles can require much more performance than skeletal meshes, if the physics for a large number of bristles have to be simulated individually. Therefore, there is always a tradeoff between performance and quality when it comes to simulation of bristles. Something which most state-of-the-art brush simulation systems have in common is that the number of bristles the system is able to simulate in real-time is low. Furthermore, while some systems are able to simulate a high number of bristles they require powerful hardware to do so. The brush simulation system used by [Chen et al., 2015] includes advanced fluid simulation algorithms, and manages to simulate between 50-200 bristles in real-time. However, the setup used includes a high end dedicated graphics card which

is many times more powerful than mobile devices are capable of [16]. The system used by [DiVerdi et al,. 2010] managed to simulate up to 100 individual bristles in real-time on consumer hardware, with higher performing systems achieving a higher number of simulated bristles. Approaches based on interpolation such as [Baxter and Lin, 2004] first calculate the physics of individual bristles and then interpolate a large number of bristles based on the simulated bristles. Their brush simulation system simulated 10 individual bristles, and then used interpolation to reach up to 200 bristles in total. The approach of interpolating a large amount of bristles instead of simulating them can give the appearance of a brush with a lot of individual bristles. However, since the shape of the brush is decided by a relatively low amount of individual control bristles, the possible shapes of a brush is still subpar. Furthermore, an interpolating approach may still have problems with distribution of bristles, such as if the bristles are distributed in two different groups or clusters, an interpolating approach could smoothly interpolate between them without regard that they are two different clusters [10].

Data driven simulations are a completely different approach. In the optimal case when requesting the state of the brush, it should be present in the deformation table. Therefore calculating a new state, it's would be nearly equivalent to the data in the deformation table. In less optimal cases, a an unknown state would be calculated from a combination of known states. With such an approach, the detail of brush is heavily affected by the data in the deformation table. In [Baxter and Govindaraju, 2010] a single bristle is used as the base for the deformation table, and thus the detail of the virtual brush is very simple. With more complex data driven models used, the more detail can be extracted from them.

2.5 Input Data

In the previous research projects such as [Baxter et al., 2001] a 6DOF input system is used where a sensors on a physical are translated into positional data for the virtual brush [17]. In [Xu et al., 2004] six degrees of freedom is required from the user in order for their system to simulate the writing process [11]. In many commercial painting applications the lowest common denominator for input is a mouse. In Adobe Photoshop, when using the mouse there are very few assumptions regarding limited user input to brush state as most parameters are set by the user in options. Changing brush properties in the middle of a stroke is very cumbersome, and leads to less dynamic brush strokes. Most mobile devices do not have very expressive data regarding user input [21], and thus there might be a need to make assumptions regarding three dimensional movement of the brush from limited input data.

Chapter 3

Method

3.1 Proposed Model

In order to implement a brush simulation system with emphasis on high detail and high performance, appropriate technologies must be used in order to achieve these properties. The algorithm strategy is probably the most impactful technology choice when implementing a brush simulation system. Procedural solutions do not rely on modeling virtual brushes, which makes it useful as a method for generating brush strokes without significant performance cost. However, the results from procedural solutions are often limited [12]. Acquisition solutions would not work well for consumer usage as they require custom hardware to implement, which makes it unsuitable for implementation on mobile devices. Touch sensitive surfaces could be used for acquisition strategies with real life brushes given high enough detail of input acquisition such as in [Greene, 1985]. However, only specific types of touch sensitive surfaces support such technologies which makes it unsuitable in this case [7].

Simulation strategies can be used for great bristle accuracy. The method used by [Diverdi et al, 2010] achieves up to a hundred individually simulated bristles with their advanced state of the method. However, it would be very difficult to implement a superior yet somewhat equivalent method within a master thesis. The brush simulation used in [Diverdi et al, 2010] proves that it's possible to implement fast bristle simulations using simulation strategies, however scaling up from a hundred bristles to a thousand bristles on low powered hardware does not seem probable. It's possible to construct a brush using simulation strategies and interpolated bristles by simulating a small number of bristles and interpolating a large number of bristles. However, this would still require the development of a highly effective simulation implementation. When extrapolating data from a initial data set to form a larger data set, it's vital that the initial data set has enough detail to be able to extrapolate from without a great loss of accuracy. For example, extrapolating five hundred bristles to a thousand bristles would probably work well while extrapolating five bristle to a thousand bristles could lead to large inaccuracies. In the initial data

set, the number of individually simulated bristles would have to be high enough to guarantee accurate extrapolation if we wish to simulate a total of thousand bristles in real-time. This might not be guaranteed to be achievable on low powered hardware, considering the state of the art solution developed by [Diverdi et al, 2010] achieves a hundred individually simulated bristles in real time.

Data driven methods show great promise in providing high performance brush simulations. Through a database of brush behaviours, data driven methods can perform simulations in constant time. Before data driven methods can be used, a database has to be populated with the underlying model. This model can potentially provide unlimited detail, and since the data is precomputed without affecting the runtime performance of the program, highly detailed simulations are possible to achieve in constant time. Data driven methods for use in high detailed and fast brush simulation systems have not been an area of much research, and thus present a perfect opportunity to research in this thesis.

3.2 Data driven model

Figure 3.1. The setup used to gather brush parameters.

The method used in this project extends the data driven model method used in [Baxter and Govindaraju, 2010], in which the brush model is based on real world data in order to reduce the cost of simulation. The data in this report was gathered by using a setup consisting of a system capable of suspending a brush in a defined position. A series of close up pictures of the brush in different positions are taken, which make up the basis for the data driven model.

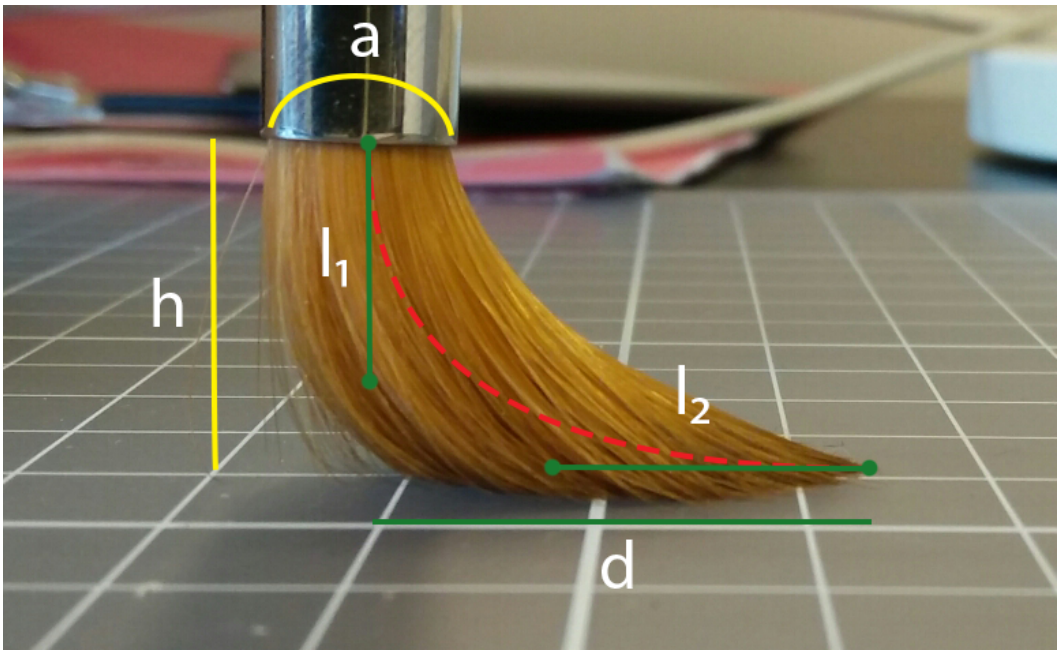

Figure 3.2. Example of the keys and values for a brush state. The key is the height and angle of the brush handle (yellow), the values of the red marked bristle is the end of the bristle to the bristle center, and the height of the second and third control points for the bezier curve (green).

A simple deformation table is created, which separates the state of the brush handle (yellow) used as key and the state of the bristles (green) used as value, as shown in figure 3.2. A key is constructed by gathering two parameters from the brush handle, the height of the brush handle to the surface h and the angle of the brush handle a. For each key, a corresponding representation of the state of the bristles is gathered, one bristle at a time.

In figure 3.2, the bristle of interest is marked with a dotted red line. The data regarding the state of the bristles is collected by quadratic bezier curves. Using image editing software, bezier curves are placed with parameters so that they overlap the bristle exactly, after which the parameters are saved as the value for the bristle state. The simplest bezier curve which would approximate a bristle consist of a starting point which is at the start of the bristle, an end point is at the end of the bristle. However, using only two data points a bezier curve would only produce a straight line which is not desirable. In order to better approximate the bristle two control points are added, which makes it the bezier curve a cubic bezier curve with four data points. To achieve this, we are using two constraints: that the line from the start of the bristle to the first control point must be parallel to the brush handle, and that the line from the end to the second control point must be parallel to the surface.

Because of these constraints, we always know the direction of the vectors leading

to the two control points. If we know the positions of the start and end of the bristle we can calculate the position the control points. This can be achieved by using two scalar values describing the distance from the start to the first control point in the l_1, and the distance between the end to the second control point in the l_2. We must also know the position of the end of the bristle, which can be calculated by adding the distance between the start of the bristle to the end of the bristle in x-axis d. With these four points, we can approximate the bezier curve for the bristle marked as the dotted red line.

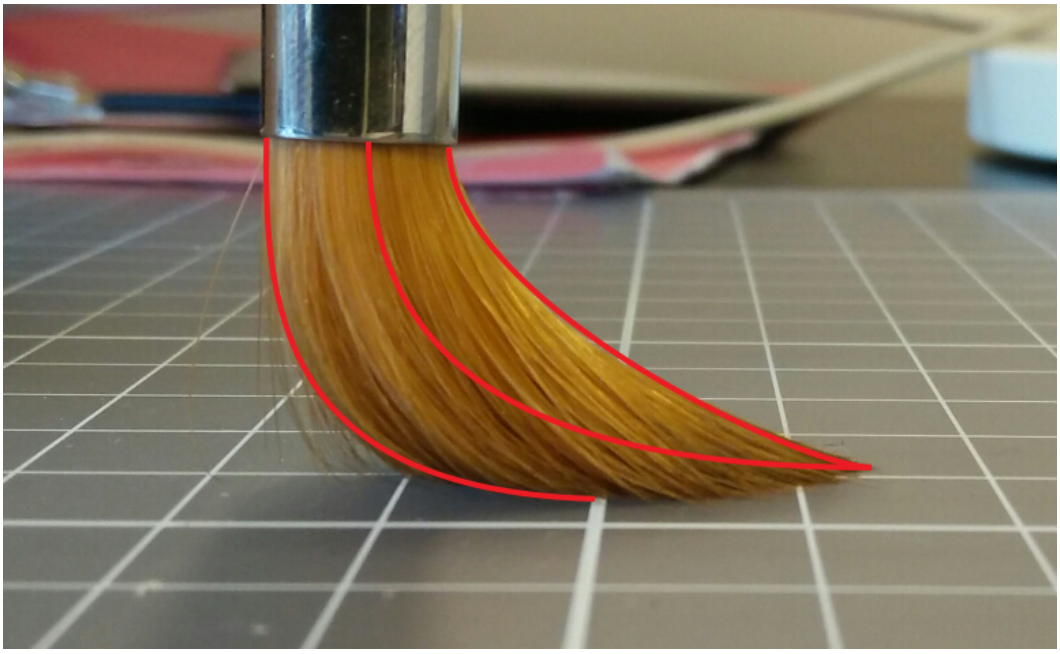

Figure 3.3. Example of the three transformations which are collected for each brush key.

For each brush key, we collect three distinct bristles together with the angular spread of the bristles in order to primitively approximate the shape of the brush. Therefore, instead of basing our model about a single line in a 2D plane such as in [Baxter and Govindaraju, 2010], we are instead approximating a very simple 3D volume. These are divided into the upper deformation, the middle deformation, and the lower deformation which describe the deformation in a cross section of the brush. The virtual brush already has a an undeformed 3D volume based on the initial properties of the 3D brush, the deformations are applied in brush cross sections where each bristles corresponds to a deformation inside the cross section in the deformation table. We exploit bristle to bristle coherence in the assumption that the deformations of bristles are related [15], and even though the deformation at the edges will be similar to the deformations in the center, the model also includes angular spread of the physical brush in the horizontal plane order to give the edges of the brush their own characteristic. Using another method which covers defor-

mation across the whole surface, and the distribution of bristles therein could yield more realistic deformations, however this method is very simple which requires very little data in order to approximate deformations and thus also requires less work in gathering the data to describe your physical brush. Gathering this data could be automated in the future however currently this data gathering is often done manually [9], the more detailed data and the more data points in the model, the more time the data takes to gather.

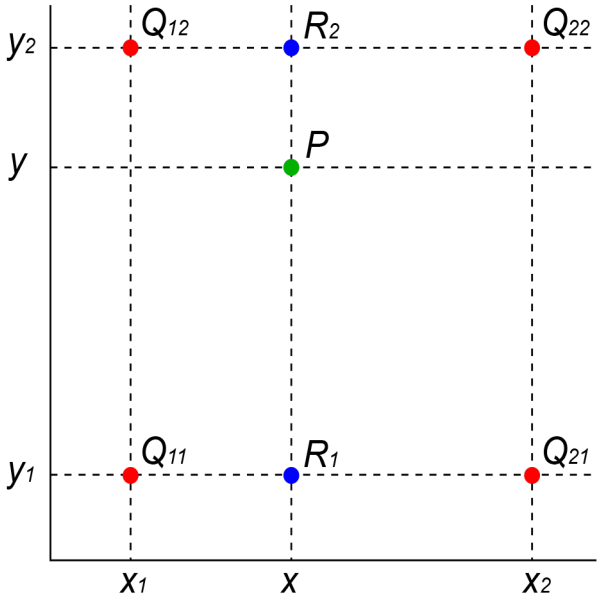

Figure 3.4. Bilinear interpolation between four distinct points.

Source: Bocsika for Wikimedia

$$
\begin{aligned}
f(x, y_1) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{11}) + \frac{x_2 - x}{x_2 - x_1} f(Q_{21}) \\
f(x, y_2) &\approx \frac{x_2 - x}{x_2 - x_1} f(Q_{12}) + \frac{x_2 - x}{x_2 - x_1} f(Q_{22}) \\
f(x, y) &\approx \frac{y_2 - y}{y_2 - y_1} f(x, y_1) + \frac{y_2 - y}{y_2 - y_1} f(x, y_2)
\end{aligned}
\tag{3.1}
$$

Figure 3.5. Algorithm of bilinear interpolation between the points Q_{11}, Q_{12}, Q_{21} and Q_{22} on a rectilinear 2D grid to approximate $f(x, y)$

In our data driven model, we are only using nine data points for brush states for the entire model, which is far below the set of all possible brush states. When we receive a new key (brush handle state), we already have knowledge of other keys relative to it, however we do do not know the value corresponding to the new key.

We can approximate the new key by using bilinear interpolation (see fig 3.5). If we name our new key P represented as the green dot, and the four red dots Q_{12}, Q_{22}, Q_{11}, Q_{21} are the nearest keys in respect to the relative direction of their dimensions as in fig 3.4.

Using the example above, we can first calculate the key R_2 or $f(x, y_1)$ by interpolating between Q_{12} and Q_{22}, and then calculate the key R_1 or $f(x, y_2)$ by interpolating between Q_{11} and Q_{21}. Since the new keys are based on existing keys with known corresponding values, we can also calculate what the corresponding values $v(R_1)$ and $v(R_2)$ would be at those new positions by interpolation. Lastly we don't need to interpolate the key P since it is are already known, however we can calculate the value of $v(P)$ by interpolating between the interpolated values $v(R_1)$ and $v(R_2)$. This is the method used for calculating the value $v(K)$ from an arbitrary key K, using it's four closest neighbours based on dimensional direction.

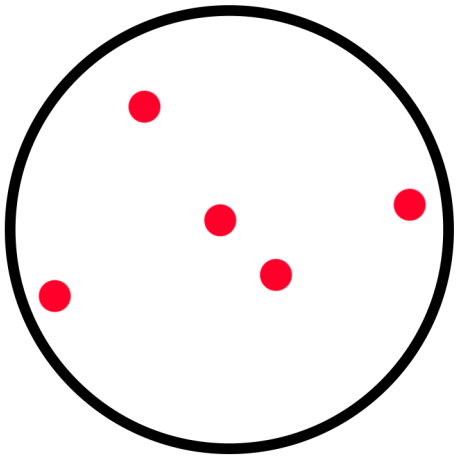

Figure 3.6. Example of initial distribution of the bristles inside a cross section of the brush. The bristles are marked in red, while the brush boundary is marked in black.

In the initial state of the brush in 3.6 the bristles are initialized according to a uniform distribution. In every step of the simulation, we have the information of the initial distribution of bristles in relation to the initial shape of the brush.

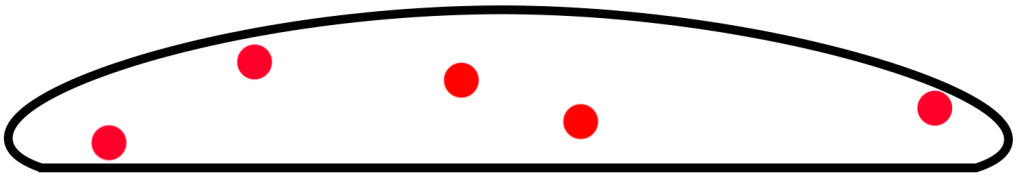

Figure 3.7. Example of distribution of bristles after a brush deformation inside a cross section of the brush. The bristles are marked in red, while the brush boundary is marked in black.

Since we are calculating the brush deformation boundaries in the brush, we can place the bristles easily as the deformed distribution is available with the initial bristle distribution and the brush deformation boundaries due to that the bristles follow bristle-to-bristle coherence [15]. This relies on the assumption that the bristle distributions remain relatively uniform, as we are using simplified data model with no brush dynamics and limited user input which limits the possible brush states. However for more advanced simulations, this would be needed to be taken into account. With this model, we have the possibility of always being able to simulate the deformations from the from the initial state of the brush. This is due to that the simulation is not dependent on any form of previous simulations, or achieving any sort of stability or convergence in our calculations as the bristle state is performed by bilinearly interpolated An advantage with a data driven system implemented here is that the next state of the brush is completely independent on the current

state of the brush, and thus the brush physics update rate could be lower than the brush imprint rate. Could yield an increase in the performance, without necessarily impacting the visual results as the state of the brush from one frame to the next are very similar. With other methods such as explicit time based integration, large time steps can easily cause numerical instability [16].

3.2.1 Model deformation from user input

For each user input, the virtual brush handle is moved, and we calculate the new bristle state. From the bristle state, we calculate the deformation of bristle vertexes according to the bristle state. This deformation takes into account the rotation and angles of the brush. Other systems such as [Baxter et al., 2001] have a physical brush handle which they track, which gives the user a one to one representation in the virtual world which corresponds to the users physical movements. We are not able to achieve a one to one brush handle representation, when the user is using their finger as input. Instead we are using a system where the users finger marks the center of the brush imprint, and in order to do this we must first estimate where the center of the imprint would be and then translate the brush to this position. This leads to a more accurate experience by having the imprint correspond to where the user is holding their finger.

3.3 GPU Computation

The data driven model very simply approximates a 3D volume. Whenever the volume is received, what's left is to distribute the bristles inside that volume according to the process described above. As we do not explicitly simulate bristle-to-bristle interactions such as [Diverdi et al, 2010], we do not need to simulate the bristles one by one in the event that the simulation of one bristle would affect the simulation of another bristle. Bristle-to-bristle simulations are already included in our model, as the bristles were affecting each other in the model brush that the measurements were taken from. Therefore, each bristle can be placed independently regarding other bristles. Furthermore, for each bristle we need to take its vertex data and then transform it to a corresponding set of vertex data after applying the bristle deformation. These two properties make this system a good candidate for performing brush physics simulations using GPU computation.

RenderScript is an Android framework for performing computationally intensive parallel work which requires high performance. It will distribute work across all available processors on a device, automating all scheduling and load balancing which might otherwise have to be written by hand. The compute kernels are written in C99, which then interface with existing Java code [23]. RenderScript is used in FastBrush to try to paralellize the physics of the simulation and the deformation of the brush.

3.4 Rendering

OpenGL, specifically OpenGL ES is used for the rendering engine in this system as it has good support in the Android framework, and fits the use case of this system. There are other fully fledged engines which provide high level abstractions and can support mobile devices, however OpenGL is a better fit since we have no immediate use for a fully fledged game engine which would only add performance overhead as we are only rendering line primitives for the brush. Furthermore, as performance is critical and most of the rendering process is based around rendering massive amounts of these line primitives, being able to modify the rendering code is crucial in order to ensure that rendering is done in an efficient way which fits brush simulation.

We have constructed our 3D virtual brush based on the data driven model using line primitives as representation. The next step is to imprint the brush onto the canvas. However, the bristle geometry directly in contact with the canvas is smaller than an imprint would be in real life, because wet brushes have a larger contact area than the geometry of the bristles [10]. An ink deposition threshold is placed slightly over the canvas in order to give a more realistic imprint surface from the brushes, as seen in fig 3.8.

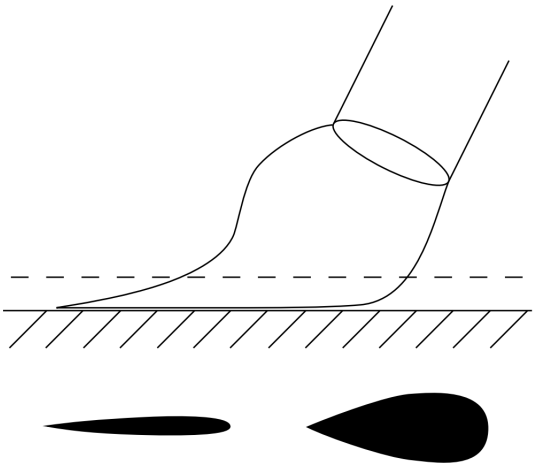

Figure 3.8. Left: The deposition of ink from the bristles directly in contact with the canvas. Right: The deposition of ink from the bristles in the deposition threshold.

Source: A Brush Stroke Synthesis Toolbox, Image and Video-Based Artistic Stylisation

The bristles are imprinted on a separate back buffer. By default, each frame is drawn by getting the frame buffer, clearing it to the background color, and then from scratch drawing the entire scene. This makes frames non-persistent, as they will be erased and rendered from scratch when rendering updated information. This method works for most cases of 3D graphics as the simulation has either strict or

loose restrictions on the upper limit of rendered objects on the screen at once, and therefore each frame can be expected to be rendered in reasonable time. However, this does not work for our case. The more a user draws on the screen, the more performance it will require to render both the previous objects and also the new objects each frame. Android fires a series of MotionEvent which detail the recorded touch input during a period of time, as fast as the device can handle. This means it's possible for an user to easily add a thousand more objects every few seconds, which is unsustainable to render cumulatively for a longer period of time. Therefore, we use the OpenGL option EGL_BUFFER_PRESERVE in order to save the previous content of the frame buffer. Instead of clearing the frame, we preserve it and only draw the new content on top. This has the obvious limitations that the visual result of previously rendered objects becomes difficult to modify, once an object has been rendered to the frame buffer the visual result cannot modified without rendering the entire frame from scratch. Each stroke is drawn on a new frame buffer, which means that it's possible to reverse strokes up to the amount of active frame buffers. For memory reasons, a ring buffer with a single digit number of active frame buffers are used in rotation which places a cap at the number of stroke undoings possible in a row. The brush is imprinted to these specific frame buffers, which are then composited together with the brush head and UI into the main buffer. This allows separation of the canvas which the user is painting on, and the current state of the program.

3.4.1 Imprint interpolation

A swipe on a touchscreen is a continuous motion which could be expressed as curves, however on Android it is represented as motion events which periodically gives a touch point. This means that when the user swipes on the screen, the data which you might receive includes the start point of the swipe, the end point of the swipe and a number of points in between depending on the speed of the swipe. Imprints must be placed no more than one pixel apart in order to guarantee a continuous stroke [13]. The touch input is not guaranteed to have a sample corresponding to every saved pixel [21], thus the implementation relies on an interpolation system which interpolates between touch points so that bristle imprints will never be more than a specific distance from each other. We also rely on the brush imprints flowing together as if they were painted in one continuous stroke, therefore we linearly interpolate or throttle all variables in touch input in order to ensure that there is an upper limit from one bristle imprint to another adjacent bristle imprint.

Chapter 4

Results

All results for FastBrush were collected on a Nexus 9 Android tablet, released in 2014 and using an NVIDIA Tegra K1 64-bit dual-core processor at 2.3 GHz. All results were collected on a canvas with the size of 2048 x 1536 pixels. Two artists have painted reference work using FastBrush, one with background in digital media, and one with background in calligraphy and physical media.

4.1 Visual results

Figure 4.1. Deformation of a brush with a thousand bristles.

Source: FastBrush

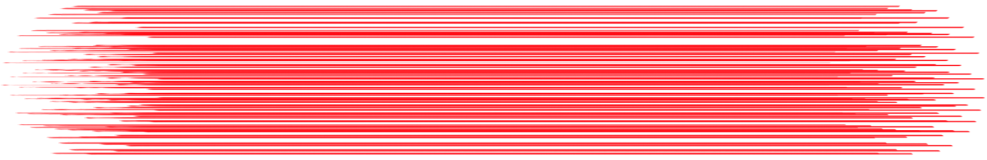

Figure 4.2. Brush simulation in Adobe Photoshop CC 2015, using the maximum brush size (300px), the maximum number of bristles (100) and the minimum bristle thickness (1%). Painted using a mouse.

Source: Adobe Photoshop CC 2015

Figure 4.3. Brush simulation in Adobe Photoshop CC 2015, using the maximum brush size (300px), the maximum number of bristles (100), and 25% bristle thickness. Painted using a mouse.

Source: Adobe Photoshop CC 2015

Figure 4.4. Brush simulation in FastBrush, using the maximum brush size, 1000 bristles and 25% bristle thickness. Painted using fingers on tablet.

Source: FastBrush

Figure 4.5. Painting made using FastBrush. Painted using fingers on tablet.

Source: Jeasmine Ljungström using FastBrush

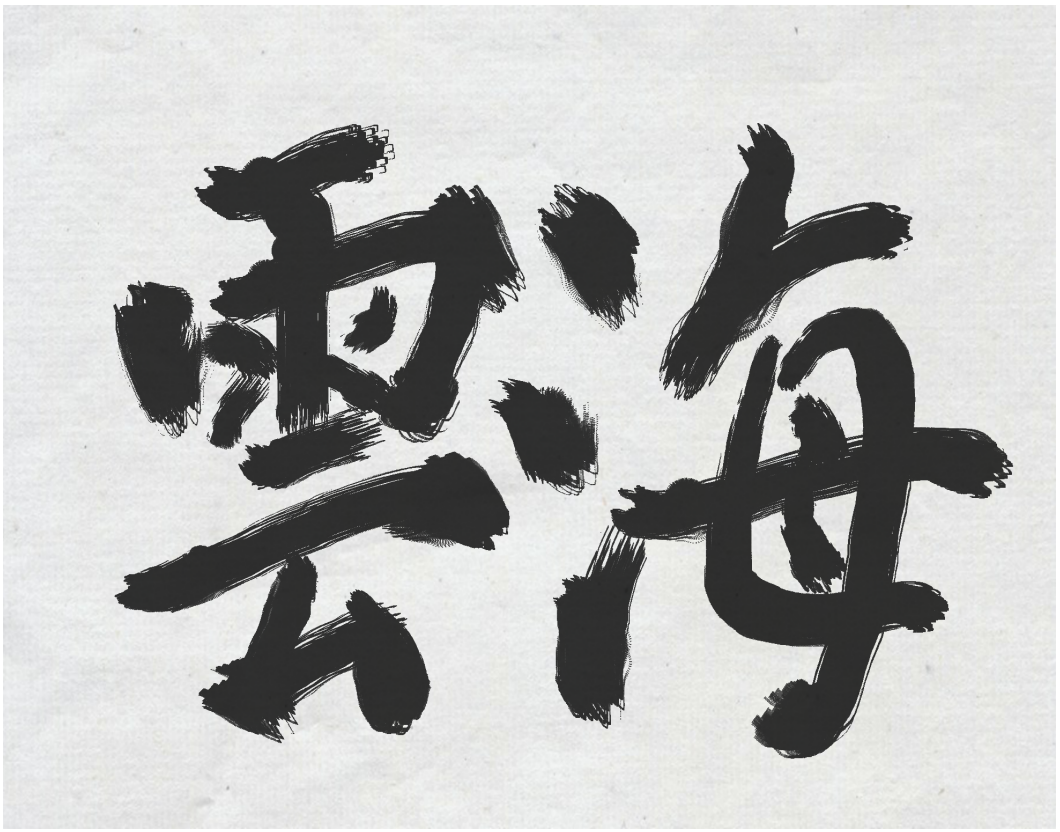

Figure 4.6. Calligraphy of the chinese word for "sea of clouds". Painted using fingers on tablet.

Source: Huiting Wang using FastBrush

4.2 Performance

When developing the application, two implementations were used for the physics simulation. The first was a single thread simulation, which performed all the simulations for each bristle sequentially. A separate implementation was later developed wherein RenderScript was utilized to parallelize the physics simulations. For each bristle a RenderScript kernel would be invoked which will be computed using available resources such as a CPU or GPU core. These two implementations have been compared in order to analyze whether single-threaded or multi-threaded physics simulations would yield higher performance in this system. The performance tests were conducted by sampling the performance of each method using one, ten, a hundred, a thousand and ten thousand bristles. The data was gathered by letting each configuration run for ten thousand iterations during real world use, and then taking the mean of the execution time. According to the AOSP documentation, the display typically refreshes at a maximum of 60Hz, therefore each frame would have roughly less than 17ms of time [22].

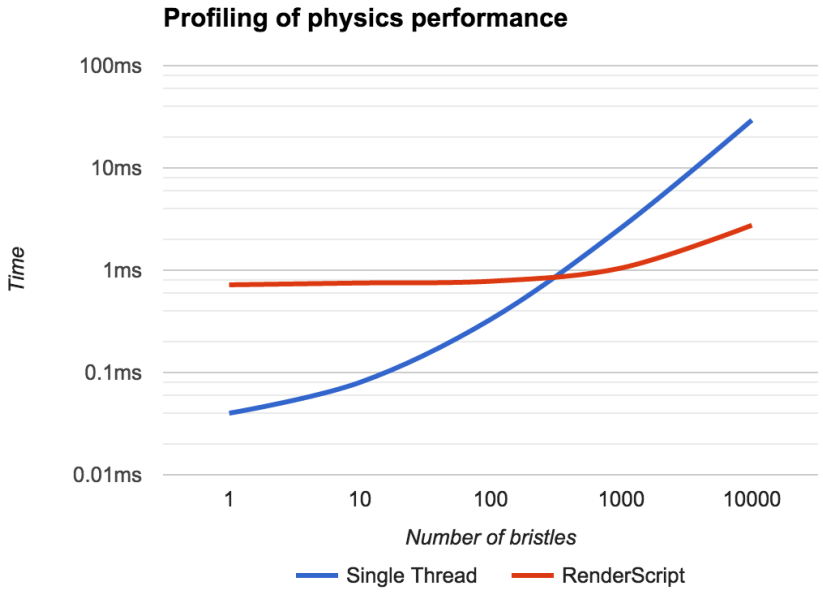

Figure 4.7. Performance of the brush physics simulation for a single imprint, with a logarithmic time axis. Lower is better.

	1 bristle	10 bristles	100 bristles	1000 bristles	10000 bristles
Single Thread	0.04ms	0.08ms	0.33ms	2.60ms	29.34ms
RenderScript	0.72ms	0.75ms	0.78ms	1.05ms	2.75ms

Table 4.1. Data of the brush physics simulation performance for a single imprint

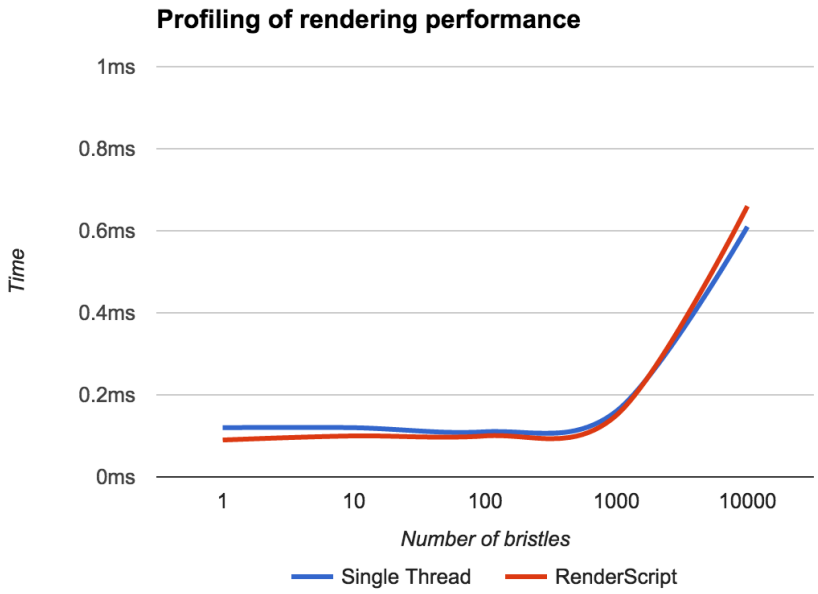

Figure 4.8. Performance of the brush rendering. Lower is better.

	1 bristle	10 bristles	100 bristles	1000 bristles	10000 bristles
Single Thread	0.12ms	0.12ms	0.11ms	0.16ms	0.61ms
RenderScript	0.09ms	0.10ms	0.10ms	0.15ms	0.66ms

Table 4.2. Data of the brush rendering performance

Chapter 5

Discussion

5.1 Visual Results

If we compare brush strokes from Adobe Photoshop in fig 4.2 to brush strokes in FastBrush in fig 4.4, we see that even though the bristle imprints in both images have roughly the same bristle thickness, the visual result from FastBrush is vastly superior in detail because of the higher amount of bristles. Photoshop uses increased bristle thickness to increase coverage when the number of bristles may not be sufficient by itself for high coverage [10]. In fig 4.3 we see a brush stroke with high enough thickness to cover the entire stroke. However, even though the coverage of the stroke is better, the quality of the ends of the stroke has been greatly reduced. In conclusion, FastBrush is able to provide far higher detail than Adobe Photoshop in regards to number of bristles and imprint fidelity. The paintings fig 4.5 and fig 4.6 have been painted by artists with vastly different styles and backgrounds, which shows that the brush simulation system can be used for a wide range of purposes. Furthermore, these paintings were painted using fingers on touchscreen, which is an input method with an accuracy far below that of mice, styluses, or custom 6DOF hardware.

5.2 Performance

From the results of the brush rendering performance of fig 4.1 and table 4.1, there are a lot of interesting observations to be made. First of all, we have to consider that the vertical time axis is displayed on a logarithmic scale because the differences in minimum and maximum runtime are several orders of magnitude apart. Thus the differences between the execution times may seem smaller when presented with a logarithmic time axis. We can see from the data, that for smaller amounts of bristles, the single core approach has a heavy advantage. It seems as though RenderScript has an overhead of roughly 0.70ms regardless of how many bristles are simulated, which makes it a very poor choice for simulating only a few number

of bristles, such as in the implementation of [Baxter and Govindaraju, 2010] where less than a handful of control bristles are used. However, we also see that when increasing the amount of bristles up to a thousand, while the computation time for the RenderScript approach stays almost constant regardless of the number of bristles, the computation for the single thread approach scales very badly, and thus at ten thousand bristles takes more than ten times the multithreaded approach. With an increasing amount of bristles, the corresponding increase in time for the multithreaded method is very low. If there would be a way to optimize the initial of the multithreaded computation, the performance could be considerably increased as at a thousand bristles it seems as roughly 70ms is spent on overhead while only roughly 30ms is spent on the physics computation. In the ends, these results show that there is possibility for parallelization using this data driven method.

From the results of the brush rendering performance of fig 4.2 and table 4.2, it's obvious that there is no meaningful difference between the rendering times between running the brush physics simulations on a single thread or using RenderScript. This is expected, as neither of the methods have any direct relation to the rendering code, however considering that RenderScript can utilize GPU computing it may be unsafe to assume that it wouldn't reduce the performance of other work performed on the GPU, such as rendering. However, in this case we can see that utilizing RenderScript has very little to no impact on rendering performance. From the aforementioned sources, we can also see that there is very little difference in the performance of the bristles rendered up to a thousand bristles for both methods, we only encounter heavy performance decreases in the physics simulation when simulating up to ten thousand bristles. Therefore the increased cost of rendering performance up to a thousand bristles can be considered negligible, as the rendering performance only truly starts to scale with the amount of bristles when they are over ten thousand, and below that cost can almost be considered constant.

The physics simulation used by [Diverdi et al., 2010] takes 30µs for a single bristle with six vertices, 30ms for a brush containing a thousand bristles which would yield an update rate of 30Hz. From fig 4.1 we see that our physics simulation takes roughly 1ms for each imprint on a brush with a thousand bristles. This is an order of magnitude performance improvement, and with the same calculations we can see that theoretically our solution could be used with an update rate of 900Hz.

5.3 Real-time Rendering

It is hard to define what counts as real-time rendering for drawing applications. For conventional computer graphics applications there is a set target rendering rate, with a known upper bound of objects to render, where each frame displays a discrete time step of the scene. With a drawing application we cannot render using the same assumptions, if we render each frame as we receive touch points, there is a large possibility that the end result will be in the form of distributed points instead of a continuous line. In order to achieve a continuous line between touch points,

touch points have to be interpolated to fill the gap between them. Rendering each touch point requires to update the physics simulation, and to perform draw calls to the GPU which are relatively expensive operations which have to be performed for every interpolated point. Therefore the time to render a new frame is directly proportionate to the distance between the touch points for a line.

If we define the average emission per second as E_s, the average consumption per second as C_s, and the length of the stroke in seconds as L, then the estimated latency in seconds for a stroke will correspond to the following equation:

$$max(\frac{(E_s - C_s) * L}{C_s}, 0) \tag{5.1}$$

Figure 5.1. Equation describing the latency of a imprinting a brush stroke

If each second has the amount of emission E_s, and amount of consumption C_s, the emission surplus will be non-negative results of $(E_s - C_s) * L$. Therefore, if E_s ever exceeds C_s, the emission surplus will grow as L grows. There are very few ways to minimize the emission surplus, if we call E_{max} the maximum number of emissions that can be yielded in a second from the two touch points with the greatest distance from each other, C_s would in theory always have to be equal or greater than E_{max} in order to guarantee no emission surplus. However, the average emission per second E_s is an order of magnitude smaller than E_{max} since the distance between touch points between frames is much smaller than the maximum distance between touch points, and optimizing C_s for E_{max} would mean losing out on a lot of performance. This means that for any system where C_s does not exceed E_{max} we can never guarantee that it will run in real-time, if we define real-time as having no emission surplus during a stroke. Therefore, as most programs optimize C_s for E_s, the standard of measurement is not real-time but rather "reasonable real-time", which in this case would mean observing whether latency is noticeable during standard usage. In [Diverdi et al., 2010] they estimate E_{max} to be roughly 500 imprints per second for typical quick strokes [10], in which case our system would pass with it's theoretical update rate of 900Hz.

5.4 Differences in target hardware

For this project, the results were gathered on a Nexus 9 released in 2014, and despite its age, it is still considered a powerful tablet with advanced graphics capabilities at the time of release [25]. Because of the used hardware being both powerful, and having a larger form-factor than most mobile devices, it is reasonable to expect that the app could perform worse on many other mobile devices. It is a deliberate decision to aim for a relatively high performance floor, which would possibly exclude a number of devices from running this app at intended quality. In [Diverdi et al, 2010] they write that one of the reasons for why they chose a hundred bristles as

the maximum amount, was because they need to target a wide range of hardware, and low end computers might not have dedicated GPUs which would introduce difficulties in rendering a high number of bristles in real-time. Android relies on GPUs for hardware acceleration to perform fundamental rendering for the OS, such as for the user interfaces [24]. The Android OS is reliant on capable GPUs for mobile devices, and thus targeting mobile devices has the advantage that while the overall performance often is no match for desktop computers, it is possible to make assumptions regarding the graphics performance on mobile devices which are not possible on desktop computers. Comparing brush simulation systems during equivalent conditions is not possible, as the benchmarks included in research papers are all performed on different hardware. Therefore, direct comparisons between brush simulation systems regarding their performance has a possibility of including a large margin of error which has to be taken into account. There is a chance that benchmarks from other papers could be improved if remeasured Nexus 9, however this is not possible due to the implementations often not being publicly available. Since our benchmarks were recorded using mobile devices, we hope that any performance advantages as a result of purely using more powerful hardware would be reduced.

5.5 Limitations and Future Work

5.5.1 Dynamics

[Baxter and Govindaraju, 2010] implemented a system for calculating the dynamics, friction, energy states of the brush. A simple data driven model is only based on real world brushes in resting states in various positions, which means that in order to calculate other external forces they have to be simulated for each bristles. Even though their system for simulating brushes is still based on the data driven simulations, they have further simulation on top of that for to take these variables into account. We have not implemented such a system, mostly because of time constraints. However, [Baxter and Govindaraju, 2010] shows that it's possible to include into a data driven system. Their system applied these calculations on the low number of bristles which were based for the brush mesh. It's possible that the cost of calculation for a high number of bristles would be very inefficient, however that is outside the scope for this implementation.

5.5.2 Data model

Since data driven brush models are based on the properties of real world brushes under deformation, the properties of the simulated brush are heavily tied to the physical properties of the physical brush used when collecting the data. In [Baxter and Govindaraju, 2010] they are able to create multiple types of different brushes. This is due to that using only a handful of control bristles as a base which are simulated individually from the data driven model. Simulating a large amount of bristles one by one from a data driven model would be possible, however this

requires an order of magnitude more model simulation than simulating a large number of bristles as a base from the data driven model. Due to that data driven methods are heavily reliant on the underlying model, the flexibility of a data driven system is only as flexible as the underlying model is. Modifying the properties of the virtual brush means that the deformation of the virtual brush is based on a different brush. Therefore it will not match the underlying model, since we are approximating the shape of the whole brush instead of a single bristle such as in [Baxter and Govindaraju, 2010]. During the brush simulations there are sometimes visible jumps between the brush transformations in certain conditions. In our data driven model, there is no enforced minimum distance between the keys for our parameters, and as such there could be two very closely related parameters which could yield large deformations based on small changes in input. Furthermore, the interpolated parameters are based on linear interpolation which can yield drastic changes. Ideally, one could precompute some sort of smoothed heightmap for all parameters which would be based on parametric curves in order to smooth out the surface instead of bilinear interpolation.

A potential point of criticism against this implementation could be since the data driven model is calculating the deformation boundaries with a high number of interpolated bristles, it loses fidelity. This is valid criticism, as all interpolation has potential loss of fidelity in case the given assumptions are not always valid. However, this method may have large benefits when compared to the commonly used interpolation where a small number of bristles are simulated and the rest of the bristles are interpolated using the simulated bristles as a base. However, that approach is heavily limited in that it's correlated to the amount of simulated bristles. With only a low amount of simulated bristles (or a heavy clustering of simulated bristles) it is possible for the interpolated bristles to be distributed in a way which does not correspond to real brushes at all. Therefore, in order to interpolated bristles correctly it requires a large and correct base of simulated bristles. This might be performance intensive to calculate and not feasible to do in real-time on devices with lower performance. The benefits with the methods used in this research, is that the deformation boundaries are looked up in constant time and are not dependent on further calculations to achieve correct results. Therefore, we are able to simulate the brush deformation with very few steps regardless of the shape of the brush or the amount of bristles in the brush. Furthermore, this is only a very simple model with few variables. More complex data driven models could yield more accurate results requiring less use of interpolation.

While [Baxter and Govindaraju, 2010] had a data driven model which fit their chosen technologies, simply extending the data driven model used by them might not be enough to work for different technologies, as their model it only approximates one bristle in a 2D plane for the whole brush. There is large potential to develop better methods to represent the brushes in 3D, in order for the data driven model to be able to fully model the brush with high fidelity. In [Diverdi et al, 2010], they criticize simple data driven models for being inaccurate for their heavy use of interpolation, for example in the case where a real brush would deform into

two distinct clusters of bristles. A simulated brush which relies on interpolation would not be able to simulate that property. In [Xu et al, 2004] they use a fully three dimensional data driven model based on control ellipses, which could show promise in further implementation. However, they use a mesh based brush which splits on pressure, which would not be sufficient for an implementation focusing on high detail of individual bristles. Another alternative method which could be used, is to simulate a large amount of bristles individually offline, and then store these results as a data driven model. This could eventually prove to improve on the weakness of simulation strategies which require expensive simulations, if the results are computed offline once and then cached for constant time access. Furthermore, it also improve on the weakness with data driven methods using large amounts of bristles, as unlike mesh based representations, there's no simple way to represent the intricacies of a highly detailed bristle brush using only simple models. This could also greatly help with increasing the number of samples, as instead of complicated maps of real world pictures of brushes to digital representations, taking the data straight from the simulation could be several orders of magnitude faster for building the model. This would mean losing the data driven part of the solution, as it would mean using an advanced caching mechanic for simulations instead of based on real world data. However, considering the difficulty of capturing the full detail of bristle brushes, such an approach could capture higher fidelity. Storing the data for thousand individual bristles would highly increase the storage and memory requirements by a thousandfold, however it seems plausible that a megabyte or two of increased memory and storage usage would not be an unreasonable criteria. An algorithm like this would most likely run slower such as many of the steps based on a simple model such as the bilinear interpolation would run a factor of thousand slower, however it remains to be seen whether there's any practical differences in performance.

5.5.3 Imprinting

For the majority of usage the imprints generated are within one pixel of each other and thus generate smooth strokes. However, there are some situations where the system generates imprints which violate this rule, which results in artifacts. These occur when the brush is pressed flat to the canvas, and the brush is moved with a high rate of horizontal angular rotation. The interpolation system is based on the delta movement at the base of the brush which means that the imprints near the base of the brush will be correct, however imprints near the edges will at time produce artifacts where the imprinting is too sparse to give the illusion of a continuous stroke. Optimally, the interpolation system would be based on the highest delta movement of any part of any bristle in order to ensure that imprints are contiguous and without artifacts. Examples of these artifacts can be noticed at the middle of fig 4.6, where it's obvious that the sides of the brush imprints have non-contiguous parts due to high brush pressure and high horizontal angular rotation.

5.5.4 Further multithreading

In [Diverdi et al, 2010] they are describing the benefits of a multiprocessing pipeline, where the simulation, rendering of the brush, and compositing of the final frame can work in parallel on different cores, so that they are not blocking the other parts. They describe that they are able to achieve up to a 2x speed increase by performing brush physics simulation on one core, and offloading the rendering and compositing to another core. In this implementation the physics simulation, rendering, and frame compositing are performed in a serial pipeline which block the other parts. Implementing a parallel pipeline shows large potential for improved simulation performance [10].

5.5.5 Awareness of ethical and social aspects and sustainability aspects

There are no ethical and social aspects and sustainability aspects worthy of being taken into consideration when creating a brush simulation system.

Chapter 6

Conclusion

The brush simulation system presented in this report achieves the set goals of simulating a highly detailed brush with up to a thousand bristles, and is fast enough to be able to run in real-time on mobile devices. The system with up to a thousand simulated bristles has higher detail in its imprints than existing state-of-the-art consumer applications, and combines high detail and high performance, which may have large applications for brush painting where real-time performance is critical. Furthermore, the brush simulation system has high potential for parallelization which makes it fit for use of physics computation on the GPU. While simple data driven models works as a proof of concept, there is a high potential for use of more complex data driven models for higher accuracy and expressiveness.

Bibliography

[1] The Artist's Handbook of Materials and Techniques
Ralph Mayer
1940

[2] Digital Paint Systems: An Anecdotal and Historical Overview
Alvy Ray Smith
2001

[3] Non-photorealistic Computer Graphics: Modeling, Rendering, and Animation
Thomas Strothotte, Stefan Schlechtweg
2002

[4] The Algorithms and Principles of Non-photorealistic Graphics
Weidong Geng
2010

[5] Hairy Brushes
Steve Strassmann
1986

[6] 3D physics-based brush model for painting
Suguru Saito, Masayuki Nakajima
1999

[7] The drawing prism: a versatile graphic input device
Richard Greene
1985

[8] FluidPaint: an Interactive Digital Painting System using Real Wet Brushes
Peter Vandoren, Luc Claesen, Tom Van Laerhoven, Johannes Taelman, Chris Raymaekers, Eddy Flerackers, Frank Van Reeth
2009

[9] Simple Data-Driven Modeling of Brushes
William Baxter, Nada Govindaraju
2010

[10] Industrial-Strength Painting with a Virtual Bristle Brush
Stephen DiVerdi, Aravind Krishnaswamy Sunil Hadap
2010

[11] Virtual Hairy Brush for Painterly Rendering
Songhua Xu, Min Tang, Francis C.M Lau, Yunhe Pan
2004

[12] A Brush Stroke Synthesis Toolbox, Image and Video-Based Artistic Stylisation
Stephen DiVerdi
2013

[13] Detail-Preserving Paint Modeling for 3D Brushes
Nelson Chu, William Baxter, Li-Yi Wei, Naga Govindaraju
2010

[14] An efficient brush model for physically-based 3D painting
Nelson Chu, Chiew-Lan Tai
2002

[15] A Versatile Interactive 3D Brush Model
William V. Baxter, Ming C. Lin
2004

[16] Wetbrush: GPU-based 3D Painting Simulation at the Bristle Level
Zhili Chen, Byungmoon Kim, Daichi Ito, Huamin Wang
2015

[17] DAB: Interactive Haptic Painting with 3D Virtual Brushes
Bill Baxter, Vincent Scheib, Ming C. Lin, Dinesh Manocha
2001

[18] Interactive Rendering Technique for Realistic Oriental Painting
Young Jung Yu, Do Hoon Lee, Young Bock Lee, Hwan Gue Cho
2003

[19] Magic Brush – A Novel Digital Painting Environment
Songhua Xu
2009

[20] Simulating Artistic Brushstrokes Using Interval Splines
Sara L. Su, Ying-Quing Xu, Heung-Yeung Shum, Falai Chen
2005

[21] MotionEvent
Android API Reference
developer.android.com/reference/android/view/MotionEvent.html
Retrieved 3rd May 2016

[22] Graphics Architecture
Android Open Source Project Documentation
source.android.com/devices/graphics/architecture.html
Retrieved 3rd May 2016

[23] RenderScript
Android Developer API Guide
developer.android.com/guide/topics/renderscript/compute.html
Retrieved 3rd May 2016

[24] Hardware Acceleration
Android Developer API Guide
developer.android.com/guide/topics/graphics/hardware-accel.html
Retrieved 9th May 2016

[25] Google Nexus 9: Preliminary Findings - GPU Performance
Joshua Ho writing for AnandTech
anandtech.com/show/8670/google-nexus-9-preliminary-findings/3
Retrieved 9th May 2016

List of Figures

List of Figures

Appendix A

Resources

GitHub Repository
github.com/adrianblp/FastBrush

FastBrush Demo Video
youtube.com/watch?v=gaym9G8vCZE