AN APPROACH TO PAINTERLY RENDERING

A Thesis

by

GARRETT STEPHEN BROUSSARD

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

MASTER OF SCIENCE

Chair of Committee,	Philip Galanter
Committee Members,	Ergun Akleman
	Stephen Caffey
	Carol LaFayette
Head of Department,	Tim McLaughlin

December 2014

Major Subject: Visualization

ABSTRACT

An often overlooked key component of 3D animations is the rendering engine. However, some rendering techniques are hard to implement or are too restrictive in terms of the imagery they can produce. The goal of this thesis is to make easy-to-use software that artists can use to create stylistic animations and that also minimizes technical constraints placed on the art.

For this project, I present a tool that allows artists to create temporally coherent, painterly animations using Autodesk Maya and Corel Painter. I then use that tool to create proof of concept animations. This new rendering technique offers artists a different avenue through which they can showcase their art and also offers certain freedoms that current computer graphics techniques lack. Accompanying this paper are some animations demonstrating possible outcomes, and they are located on the Texas A&M online library catalog system.

The painting system used for this project expands upon an algorithm designed by Barbara Meier of the Disney Research Group that involves spreading particles across a surface and using those particles to define brush strokes. The first step is to infer the general syntax of Painter's commands by using Painter and its ability to record a painting made by an artist. The next step is to use the commands and syntax that Painter uses in the automated creation of scripts to generate paintings used for the animation. As this thesis is designed to showcase a rendering technique, I found animations made by fellow candidates for the Master of Science and Master of Fine Arts degrees in

Visualization bearing qualities accented by a painterly treatment and rendered them using this technique.

DEDICATION

This thesis is dedicated to my family, Dad, Mom, Taylor and Lauren. I also dedicate this project to every artist that dreams about changing the world of art and movies.

ACKNOWLEDGEMENTS

This thesis would not exist without the faculty, staff and students of the Visualization Lab. The creative students of the lab have created the artwork used for the renderings, including Krista Murphy, Christine Li, John Pettingill and the Sleddin' Team. My classmates showed genuine interest in this idea that reminded me why I want to make art and offered amazing feedback along the way that kept me going. The faculty offered help whenever it was requested and are a primary reason this research was a success. The members of the staff are the ones behind the curtain that keep the Viz Lab going and never request appreciation; you taught me so much in such a short time.

Thank you to my father for his support and guidance, to my mother for her inspiration, to my brother for friendly competition, and to my sister for making me feel like a movie star. Thank you to Krista who showed me unconditional support, kept me focused on my goals, and never let me stay satisfied with less than my best.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER I

INTRODUCTION

Most audience members that watch animated movies do not know that they have watched “rendered” frames or that the visuals were produced by a combination of mathematics and art. The highly representational aesthetic style of those animations is referred to as photorealistic rendering. Most artists working in the field are unaware that non-photorealistic rendering, or NPR, is a highly achievable method of displaying animation that can allow the artist a very high degree of control and freedom. With this project I present software that allows artists this very high level of control through a subset of NPR called painterly rendering. The term “painterly” means that the software renderer simulates a stylized look as if the animation was painted by a human. While most contemporary animations strive for realism, this renderer aims for a gestural and abstract look. This tool allows the artist to be able to take animation data from a commercial software package that typically strives for photorealism, and renders out the animation in this “painterly” style.

I.1 Introduction

Modern artists have incorporated the speed and flexibility of computers into the discipline of painting in a process referred to as digital painting. An artist can now paint on virtual canvases of arbitrary size, and apply and undo brushstrokes without

considering the cost of canvas or paint. Currently, computers can simulate complex interactions between media and allow for actions that are impossible in the physical world, such as the application of a rainbow brush that changes color throughout the application of a single stroke. I have chosen to take advantage of this combination of traditional art and computer programming as my rendering platform.

One of the issues associated with this combination is the rapid change in information from frame to frame, or a lack of temporal coherence. According to a report on current techniques designed to enhance temporal coherence for stylized animations (Bénard et al. 2011), the quality of temporal coherence can be quantified in three ways: perceived flatness of the image, motion coherence and temporal continuity.

Flatness of an image is achieved when the presented image looks hand-drawn rather than applied to a 3D surface. To demonstrate this, imagine holding an apple in your hand and rotating it slowly. A flat animation would be created by drawing every frame of the rotation by hand while watching the apple. An animation with depth would be created by painting on the apple and recording the rotation with a video camera. Both animations would be painterly but the latter would not look flat, or hand painted, while the first would flicker from frame to frame. A third approach to painterly rendering involves repeating brush strokes of the same size in the same position for each frame but only changing color values. This is referred to as the shower door effect (Meier 1996) - named so because the objects look like they are moving behind a glass shower door. The first and third techniques produce flat images but both lack one of the other two qualities.

Motion coherence is correlation between the apparent motion of the brush strokes and the motion of the objects in the 3D scene (Bénard et al. 2011). An animation using static brush strokes demonstrates the shower door effect and thus lacks motion coherence. Another example lacking in motion coherence is an animation where the objects are static but the brush strokes move regardless. While both of these effects can be interesting and appealing, they do not meet the goals made for this project. The best way to avoid this problem is to directly link every brush stroke to an object in the scene so that its position and size change with the geometry in the scene. This link, however, can cause animations to lack temporal continuity.

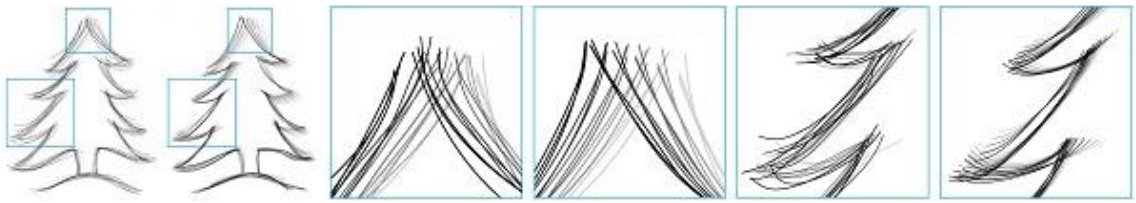

Fig. 1 - Temporal Coherence (Noris et al. 2011)

Temporal continuity is the changes in positions of the brush strokes from frame to frame that are unrelated to changes in the virtual 3D scene. Fig. 1 shows overlaid frames from two hand drawn animations. The leftmost tree is drawn differently from frame to frame and would flicker if played in motion but the rightmost tree would be temporally coherent. Referring back to the apple example, the hand painted animation of the rotating apple would lack temporal continuity because no human could perfectly match brush strokes between frames. The particle system introduced by Meier (1996)

minimizes flickering of strokes because each particle retains 3D positional information between frames but some flickering is still introduced when determining the order in which to draw the particles. If the camera moves, the particles changes distance to the camera thus changing the drawing order and causing the animation to flicker.

Bénard et al. (2011) classifies current techniques as either few-marks methods or many-marks methods. As implied by the name, few-marks methods produce images with as few marks as possible. These methods reduce clutter in images and help maintain flat images but usually result in poor temporal continuity due to brush strokes appearing and disappearing. Some techniques also move brush strokes across the surface of a 3D object to try to cover holes in the image. Many-marks methods attempt to minimize noise by painting a large number of brush strokes; so many that popping of individual strokes is hardly noticeable. The problem with this method is that the placement of the strokes is based on noise patterns, which can be seen in the produced images. Fig. 2 shows an example of the few-marks technique designed by Vanderhaeghe et al. (2007) and Fig. 3 shows a many-marks example from Benard et al. (2011).

Fig. 2 - Few-Marks Rendering

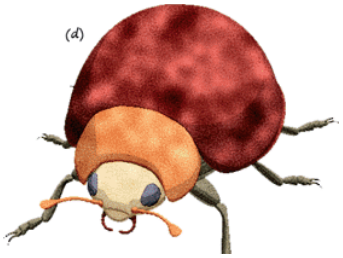

Fig. 3 - Many-Marks Rendering

However, there are other problems to solve in the field of painterly rendering. A report regarding current painterly rendering techniques (Hedge et al. 2013) splits painterly rendering techniques into two stages: low-level simulation of physical paint properties and individual paint stroke generation. Fig. 4 below shows how Hedge et al. (2013) chose to divide up the research in the field of painterly rendering.

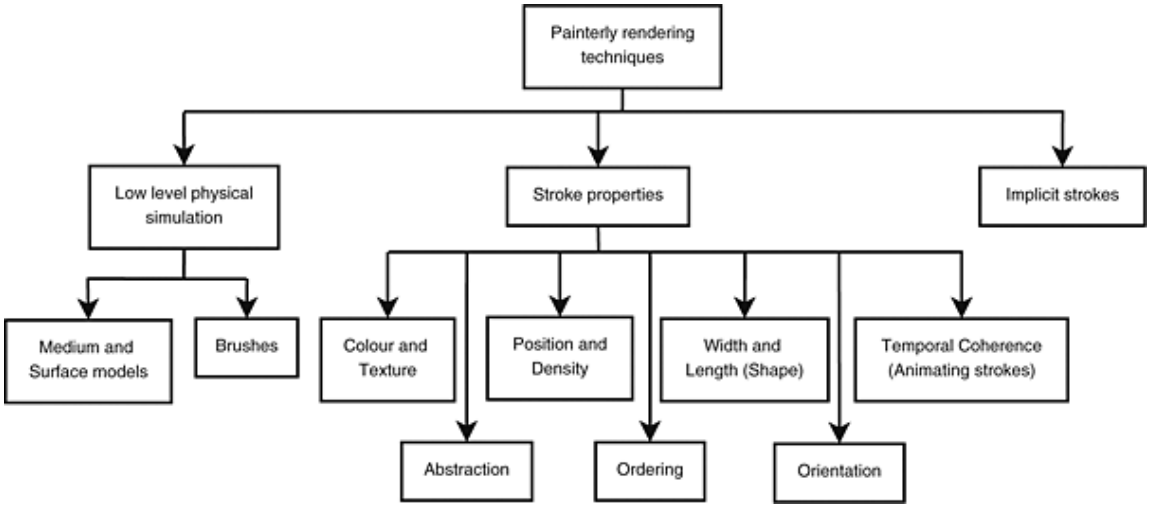

Fig. 4 - Painterly Rendering Hierarchy

Physically accurate paint brush simulation is a popular topic of research and no review of current techniques would be complete without an overview of *Hairy Brushes* (Strassmann 1986). *Hairy Brushes* was one of the first attempts to simulate the bristles in a paint brush instead of stamping images of brush strokes. However, the strokes were defined by a list of points that held time and pressure values, which made it very unfriendly to artists While this paper had limitations, it interested researchers in the interactions of bristles with neighboring bristles, ink and the paper to which the ink was applied. Followers of Strassmann, including Lee et al. (1999) and Baxter et al. (2001), continued to develop more intricate systems for simulating the application of a paint brush to paper. Lee et al. (1999) adapted the paint brush simulation to run in real time but failed to add color or a better user interface. Baxter et al. (2001) created a haptic interface that gave the user a feeling of holding and applying a paint brush to paper. The algorithm allowed for colored inks and accounted for ink already applied to the canvas,

taking a large step forward for the field. However, it was not until Zwicker et al. (2002) created a system for painting on point-based models that the idea proved to be useful for painterly animations. Once a technique was designed to simulate application of a brush to paper or a 3D model, a system was needed to control the higher level attributes of the brush strokes, such as position, shape and color.

Some methods of brush stroke generation require abstracting shapes, or reducing details in the shape, to create a painterly feel. Papari et al. (2007) proposed a method that reduced texture details but preserved the edges of the object. *Artistic Vision* (Gooch et al. 2002) used depth information to control the level of reduction. These methods are designed to turn 2D images into paintings but could be extended to 3D renderings in terms of abstracting objects in the background and drawing attention to the foreground. In order to determine that the placement of brush strokes sufficiently covered a surface, some researchers made artists place all strokes by hand and generate orientations for those strokes (Haeberli 1990). Others generated random values for positions on the surface and placed strokes where the value exceeded a threshold (Park et al. 2008). This resulted in uneven distributions across the surface and allowed for little artistic control. Hertzmann (2001) moved existing strokes to fill in gaps using relaxation algorithms to evenly distribute strokes across the surface and in so doing reduced temporal continuity. Kowalski et al. (1999) randomly assigned a user specified number of strokes to an object and then allowed the user to move those strokes across the surface to ensure even distribution. This is an even blend between artistic control and efficient use of an artist's time. Many techniques have been designed to create painterly renderings that vary from

complete artist interaction to complete automation; the techniques covered in this section are only a small selection of the research done in this field.

I.1.1 Definitions

Some portions of this paper contain terms that refer to digital art, including animations that are hand drawn using a stylus and 3D animations, while others contain terms related to physical art media. Some overlap exists between the two and I will be clear about any conflicting definitions between the two disciplines. It is not a requirement to be an expert in this field to understand this paper, but some terms must be understood before proceeding.

Mathematically, rendering is defined as a camera in 3D space with a 2D plane attached that holds the projections of the objects in the scene. This process requires linear algebra techniques as well as state of the art computational hardware and processing efficiency. This process is the most expensive part of the modern animation process. Expense here refers to the amount of computer processing time necessary to complete a task. More detail is given throughout the paper regarding rendering techniques and implementations.

A digital painting is a painting that is created on a computer using a digital software package, such as Painter (Cowpland 2012) These paintings can be quite photorealistic and approximate paintings created on a physical medium, such as oil on canvas. Digital tablets with pens resemble physical brushes and media, creating a

feeling of physical painting. Tablets are preferred over mice because of they are pressure sensitive and give haptic responses similar to painting with a brush.

Painterly rendering is a style ofnon-photorealistic rendering, orNPR, that makes rendered frames look like moving digital paintings. The two common approaches to achieving this style are to automate 2D digital paintings or simulate the paintings with 3D imagery. The difference between these two approaches is that the 2D paintings are not rendered from 3D geometry and rely solely on image manipulation. The 3D imagery involves processing 3D geometry and attaching brush strokes to that geometry. Painterly here refers to images possessing a style that resembles an actual painting, regardless of the level of realism. A painterly image can show lighting interactions that are impossible in the real world or distort the viewing angle to create abstract forms, an ability that photography lacks.

A frame is a single image, and if a series of frames are played at a rate of at least 24 frames per second, the viewer can believe the frames show continuous motion (Watson 1985). This is directly tied to the concept of rendering and also helps link this process to physical painting with one frame being one digital painting.

3D describes a space defined by three axes, or dimensions. The world we live in is described as such, whether it is x, y and z or height, width and depth. 2D describes a plane that exists in only x and y. Computer monitors and television screens are the most common examples of this space. 2D lacks the physical depth component so any perception of depth is an illusion created by an artist; this is important to understand for the remainder of the paper. In “analog” images such as drawings on paper and paintings

on canvas, the illusion of depth is generated through the use of various types of perspective, through layering and through the effects of light and shadow. On a digital display, the illusion of depth derives from the techniques used by analog images, stereoscopic imagery and the motion of objects on screen. This will be the only mention of stereoscopy in this paper. 3D will only refer to digital animation techniques.

The "industry" refers to the animation industry focused on creating 3D feature length animations. This industry also encompasses gaming and visual effects companies, but 3D animation for feature length animated films is the main focus for this paper.

I.1.2 3D Versus 2D

Animators choose to animate in 3D because of the consistency that the medium provides, but also because an object created in 3D can be shown from all angles without having to be recreated, unlike hand-drawn or 2D objects. 2D animation is created by drawing every frame by hand, usually from reference imagery. The objects in 2D animations are flat and do not offer multiple angles, so rotating a character from a 2D animation would be similar to looking at a photograph from the side - you would have no new information to look at and the image would be distorted. An object in a 3D animation has virtual depth and can be seen from any angle. This is an advantage because 3D objects can interact with new light sources, which would require the artist to create an entirely new 2D object. The downside of 3D animation is that it eventually has to be displayed on a 2D monitor or screen, or "rendered out". As mentioned in the

definitions section, rendering is expensive and requires more computation for less noisy results. Image quality in a 2D animation is based on the drawing skills of the animator.

2D animation has a major drawback, this being the flickering, or popping that happens when an artist draws one frame slightly different from the previous frame. This is the issue of temporal continuity referred to in the introduction (Fig. 1).

I.2 Motivation

Artistic styles are often followed by styles that are purposefully different than their predecessors, which I will refer to as counter-styles. For example, formless, floating subjects have often been countered by photorealistic and physically accurate styles that followed. Abstraction and casual styles are sometimes in direct opposition to methodical and mechanical styles. The last of the two examples bears the most relevance to the project. I feel that this new painterly rendering style proves to be a prolific counter style to that of the current animation industry. In my opinion, animated film audiences have been saturated with a mechanical and formulaic animation style that is based on focus groups and box office potential so much that they deserve a new original style. I propose an alternative to mechanical photo-realism by combining current rendering techniques and technology with the style of Impressionistic art using digital painting software.

I.2.1 A New Avenue

After 40+ years of research and innovation, computer graphics engineers have developed standards of programming style and practice, but some limitations still remain. These standards are important to consider because they are an obstacle that keeps studios from changing style from film to film. Studios invest money into the software needed to make their films, so they choose to invest in software that has proved to be effective and hesitate to try unproven tools. Although current rendering methods produce a limited style of art, they have been proven to be computationally efficient and the style brings people to the theater to watch these movies. However, this current generic style leaves much to be desired. The generic style can be described as animations with a high level of realism and detailed textures. While some animations are more stylized than others, they are still bound by the ideas of photorealistic light interaction and therefore lack artistic options. The photorealistic style does not allow for physically inaccurate depictions that artists sometimes use to express an idea. My opinion of the movie making process is this: the mechanical animated style was developed through research done by large studios in the form of focus groups and audience reactions. Focus groups are used to predict how the audience will react to a movie once it is released. The focus groups are shown the most recent version of the film and asked to answer questions regarding the plot of the movie, character appeal and overall film quality. The film is then changed based on these critiques. Studios judge the success of a film partly on the revenue generated at the box office. If a film does poorly at the box office, it is

dissected to find the points of failure. Sadly, deviations from the formulaic style are usually chosen to be the reason for failure and removed from future films. It is this fear of box office disasters that restricts artistic advances from making it to the box office.

I.2.2 Artistic Freedom

As mentioned earlier in the introduction, artistic choice is a requirement for a rendering technique to be functional and successful. If the technique displays an image on the screen that has uncontrollable noise, or the artist cannot control the look of the image, the technique lacks utility. A common error found in animations is a slight difference in pixel color from frame to frame that causes a flickering, this is called noise. Beyond minimizing noise, this renderer should allow artists to produce a variety of image styles with any look that is desired, which is what drove me to design my rendering technique. An artist can choose short and choppy brushstrokes or long and elegant ones. The artist can even paint with multiple types of media by simply changing a value within the renderer.

A goal for this project is to not require the artist to learn a new piece of software every time a new painting style is explored. The only learning that should happen is the artist experimenting with these different styles, just like a physical painter would learn a new style. Technology is required for this style of art but should not be a burden to the artist if it can be avoided. I believe this technique addresses that issue because of the wide range of options and ease of use.

CHAPTER II

CURRENT TECHNIQUES

II.1 Evolution of Rendering

Early methods of rendering involved an approach where objects are projected onto the camera plane and drawn over by other objects closer to the camera; this is the Painter's Algorithm (Newell et al.). Modern forms of rendering involve shooting a "ray" from the camera and finding the nearest intersection point with an object, ray tracing (Kajiya). Ray tracing allows for faster rendering times because not every object needs to be drawn. Ray tracing has evolved and allowed for extremely complex approximations of real-world phenomena, such as scattering of light inside a surface (Hanrahan and Krueger, "Subsurface") and even bouncing color between objects that are close to each other (Hanrahan et al., "Radiosity"). This has allowed the industry to push harder and harder for a "photo-realistic" aesthetic but still falls short of experiencing an object in real space. As a result, most 3D animations have a singular, formulaic feel, which Meier referred to as a "mechanical look" (Meier). However, if a tool existed that provided artists with a way to change the overall look with each new animation it would allow the story to dictate the art and have the look match the feel of the story. Some studios, such as Laika, have developed a way to do just that with stop motion animation but none have solidified a technique for 3D animation. Imagine if the concept art for *Frozen* (Fig. 5) turned into the final look rather than just a distant source of inspiration.

Fig. 5 - *Frozen* Concept Art

II.2 Current Painterly Rendering Techniques

There are many problems to solve when creating a painterly animation, the biggest being temporal coherence. Modern ray tracing applies a certain amount of randomness to each ray to avoid undesirable effects in the image that draw attention to the image being digital.

Unfortunately for painterly animation, the issue of coherence goes far beyond flickering and is a major deterrent for using this type of rendering on a feature film. Hand drawn animations usually show popping and flickering from frame to frame because the artist did not draw each frame exactly the same as the one before. The classic Disney movies and older Looney Toons cartoons minimized the flickering by using a technique called rotoscoping (Maltin 1980). Rotoscoping involves filming a

person performing an action and then tracing over the movie frames to draw a digital character performing the same action. This technique made the motion more fluent, but silhouettes and line quality would still pop from frame to frame. Quite a few contemporary techniques suffer from this problem because the brush strokes exist in a 2D plane,the camera plane. For the purposes of this thesis, I chose to follow Barbara Meier's *Painterly Rendering for Animation* technique because the 3D brush strokes naturally cohere from frame to frame. The problem with this technique is that using Newell's Painter's algorithm needs more computation for rendering when objects are added or the length of the animation is expanded. Naturally, this would encourage one to use the contemporary technique of ray-tracing, and in doing so minimize the number of strokes that are drawn. The problem with this approach is that the randomized behavior of a ray shot at a static object could easily hit different strokes from frame to frame, leading to popping and flickering.

As this thesis demonstrates, the solution lies in a technique that samples brush strokes based on camera position but maintains the information between frames. The information in this case is the relative position of the paint strokes on the surface of the geometry. The following sections are broken into techniques that use 3D information, geometry-based techniques, and techniques that use pixel information, pixel-based techniques, to create brush strokes. In the context of rendering animations, geometry-based approaches render paintings directly from 3D geometry while pixel-based approaches use traditional rendering techniques to generate an image and then create a

painting from the rendered images. Only the geometry-based approaches have proven to be temporally coherent for animation.

II.2.1 Geometry-Based Techniques

The idea of non-photo-realistic rendering has existed for over two decades and the movement has produced a variety of techniques. The technique used in this paper was based on *Painterly Rendering for Animation* (Meier 1996). The paper's breakthrough was the idea to attach brush strokes to the surface of objects using particles in 3D space. When the brush strokes exist in 3D virtual space, they naturally and smoothly exist from frame to frame with few coherency issues. These particles contained data that controls the color, paint brush type, size, etc. of the brush stroke to be drawn at render time. Fig. 6 shows a painting from the paper that was inspired by Monet's Haystack Series.

Fig. 6 - Barbara Meier's Haystack

"Landscapes of Color", Davis (2011), expanded on *Painterly Rendering for Animation*, but chose to create the brush strokes in 3D space and rendered them with Renderman, Pixar's proprietary renderer. This allowed Barrett to see the paint from different angles and even create effects like occlusion and shadows on the paint. Davis aimed to create digital paintings in the sense that the paint exists in analog 3D space. The audience could move around the painting and see the layers of paint; they could also move lights around the painting and watch the interaction. Davis' technique was interesting, but created forms that were too abstract for storytelling. This technique could prove to be very beautiful to look at in motion if it provided a high level of artistic control. Fig. 7 shows one of Davis' paintings from his paper.

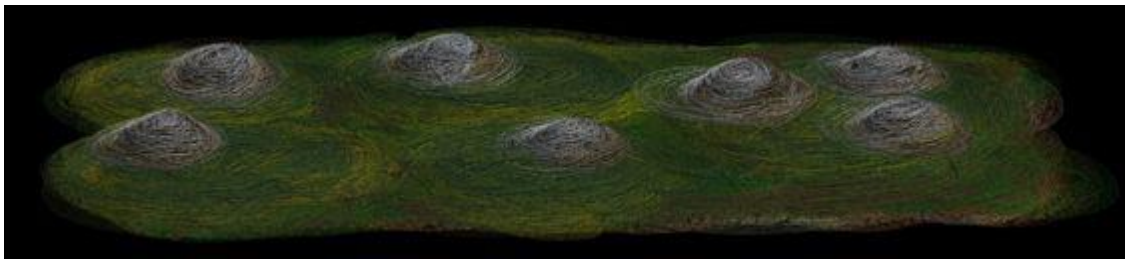

Fig. 7 - Landscapes of Color

Another technique is Deep Canvas, a tool created by Eric Daniels of Walt Disney Animation Studios and used in the movie Tarzan. Deep Canvas stored brush strokes in space but also allowed the artist to paint directly onto a virtual 3D model, a process automated by Meier. This approach allowed for more artistic direction and precision in the look and placement of the brush strokes. Artists had complete control over how the paintings looked, rather than relying on an automatic process. Deep Canvas is the only technique mentioned that was actually used in a feature length animated movie. Deep

Canvas was used in the tree surfing scene in *Tarzan*, shown below in Fig. 8. The use of this software in the movie *Tarzan* won Daniels an Oscar for technical achievement in 2003.

Fig. 8 - Deep Canvas

A contemporary technique from Pixar's Michael Kass known as the temporally coherent Image Analogies algorithm (TCIA) used 2D “texture synthesis” and keyframe paintings from artists to interpolate between frames (Kass et al.). This allowed artists to easily and intuitively create painterly animations quickly, with little time needed to learn the software. When the software was given a lit 3D scene and 2D keyframe paintings, it was able to interpolate between the keyframes and synthesize textures that appear throughout the animation by blending between these keyframes. The software even handled objects that become occluded and disoccluded, where occlusion refers to an object being hidden behind another object. The software was used to create an

animation of an ice skater that was played at SIGGRAPH 2013 (Benard et al. 2013). Fig. 9 shows some keyframes used to create the animation.

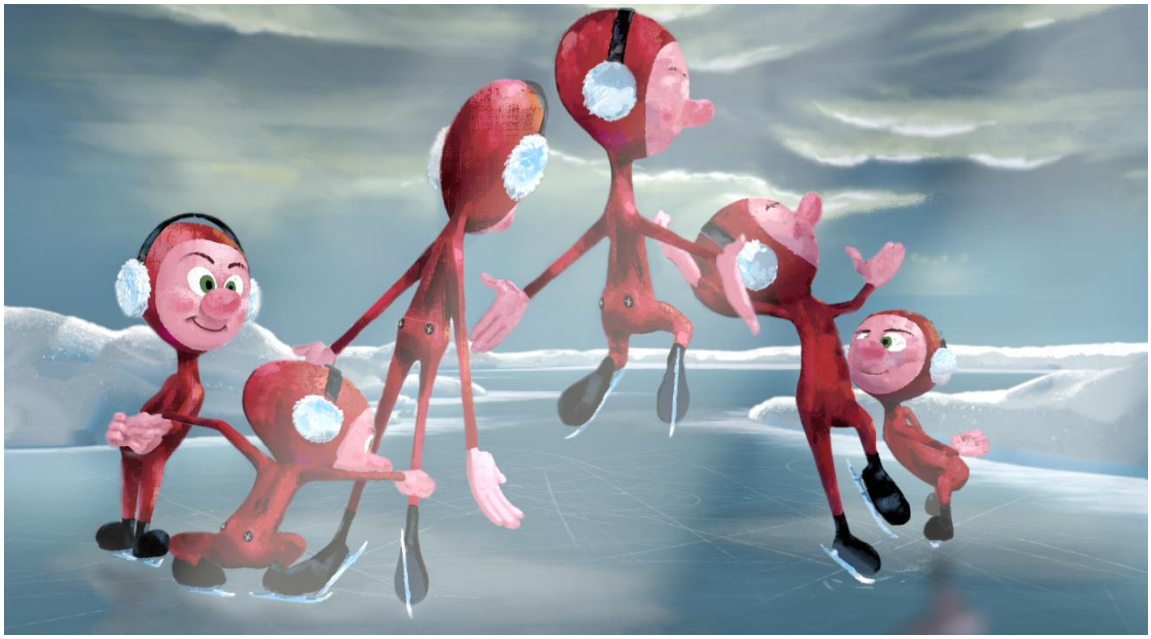

Fig. 9 - Stylizing by Example

Markosian et al. (2000) presented an algorithm to render stylistic animations with minimal levels of detail. The algorithm created procedural textures, graftals, which generated the level of detail needed at render time. The level of detail of each graftal was controlled by a tuft that has properties set by an artist. The graftals were textures applied to a surface that caused the textures to distort if viewed from an extreme angle. This approach worked here given the minimal amount of detail in the animations. Fig. 10 features an image from the paper showing the Dr. Suess-like quality of the imagery.

Fig. 10 - Graftal Based Animation

II.2.2 Pixel-Based Techniques

Artistic Vision (Gooch et al. 2002) generated a painting from a processed image using computer vision techniques. *Artistic Vision* took an image in as input and segmented the image based on gray scale values. The segments were cleaned up using a hole-filling algorithm. Once the segments were created, a brush path was given to each segment and the segments were rendered using those paths. The placement of the brushstrokes was based on the medial axis of the segments gathered from the image processing. Gooch referred to the medial axis as the skeleton of the segment. This technique produced some beautiful imagery, was quite user friendly and could be controlled artistically, but has not been proven to be temporally coherent. There could be a possible combination of this technique with *Painterly Rendering for Animation*, where *Artistic Vision* takes in

texture maps from artists to create brush stroke patterns and the strokes are stuck the surface as particles then rendered out. Fig. 11 shows a puppy playing in the snow; the pink spots demonstrate the underpainting made by the software.

Fig. 11 - Artistic Vision Rendering

Interactive Vector Fields for Painterly Rendering (Olsen et al 2005) used semi-Lagrangian fluid flow dynamics to create expressionistic painterly animations. This means that the brush strokes resemble objects stuck in a turbulent body of water. Based on the paper, the technique was effective and easy to use but, again, cannot be used to render 3D animations. The idea of fluid flow driving brush stroke orientation could be

incorporated into a technique created for painterly animation. Fig. 12 shows an expressionistic painting made from a photograph of a sky over trees at sunset.

Fig. 12 - Interactive Vector Fields

A technique from Zeng et al. (2009) rendered paintings from images but did so in an object oriented fashion, meaning that the painting not only had knowledge of which strokes made up the same object, but whether the objects were human faces, road signs, trees, etc. An artist can use different brush types and stroke types based on the type of object being painted, so an automated process should do the same. The ideas presented

by Zeng et al. were very exciting for the field of pixel-based painterly rendering, but 3D animations already contain this semantic information and this information can be changed by the artist easily. Fig. 13 shows the orientation field for brush strokes after the image has been split into semantic objects.

Fig. 13 - Object Based Orientation Field

II.2.3 Implementation of Techniques

I have implemented a system quite similar to Meier's, but with the 3D interaction available from Deep Canvas. I read in object information from a 3D animation, attached particles to the surface and displayed those brushstrokes in real-time using OpenGL, an

open source graphics library. Once the artist was satisfied with the real-time preview of the painting the scene was written out as a "recording" to be played back in Painter and saved out as frames for the animation. Unfortunately, the use of Painter could result in a processing bottle-neck for the system as each render had to be initiated by a user and could not be split among multiple machines, known as batch processing. This is discussed in length in the following section.

CHAPTER III

OBSERVATIONS

III.1 Digital Paintings

In my experience, Painter mimics physical properties of instruments such as pens, chalk, paintbrushes, etc. and substrates such as various types of paper and canvas, while allowing artists to work digitally. This feature set gives the software a unique set of pros and cons that are not present with current rendering packages. Downsides to this software include the amount of interaction needed to perform tasks and the slow response and processing times of the program. Painter was designed to be an interactive program, so nothing can be done through a batch process. Also, Painter was only optimized to perform at a rate expected to keep up with a human painter and is slower than programs designed by Strassman (1986) and successors. This can cause render times of two hours and thirty minutes per frame, an average render time for a studio with a collection of render machines but an extremely long time for a process that it so interactive. Overall, the downsides are outweighed by the versatility and functionality of the software and its ability to make a digital painting look as though it were created in the physical world. The paint simulations produce the illusion of physical accuracy and the illusion of impasto in a digital painting produced with the program can even be lit in a way that virtually approximates the method in which an artist would present work in a gallery.

III.1.1 Hand Paintings

I studied my own paintings, as well as other artists' works, to learn about the painting process so that I could make a tool that was artist-friendly. Fig. 14 shows a painting made in Painter by Master of Science in Visualization candidate Laura Murphy. Please note that the brush strokes follow the form of the 3D model and that the position of the light is shown by the use of highlights on the jellyfish. Fig. 14 uses dramatic lighting and brush stroke direction to express the form of the jellyfish. These concepts inspired me to calculate light contributions for each particle as well as the need to orient the brush stroke along the surface. Fig. 15 and 16 show a close up view of brush strokes created in Corel Painter. The first image shows a single brush stroke and the second shows three brush strokes interacting with each other.

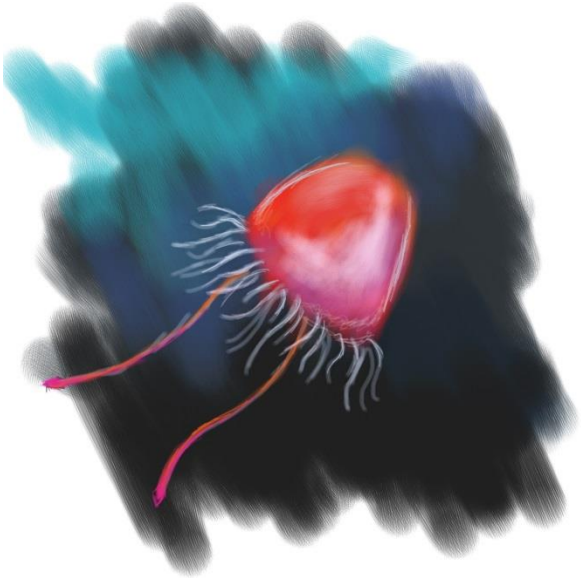

Fig. 14 - Laura Murphy Painting

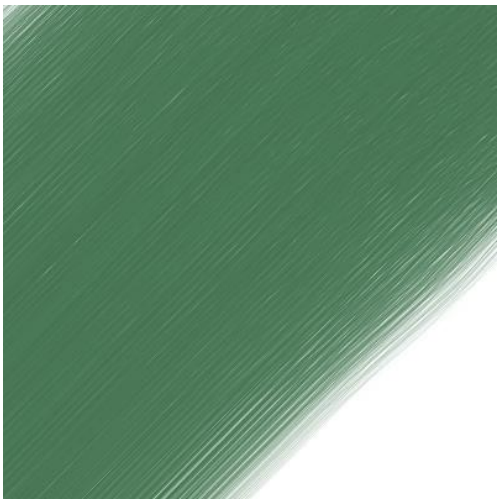

Fig. 15 – Brush Stroke 1

Fig. 16 – Brush Stroke 2

III.1.2 Automated Paintings

After creating a complex system for painterly rendering, it was necessary to test this system to ensure that it produced results that met aesthetic criteria of strong visual appeal and painterly resemblance. Figs. 17 through 19 show test paintings of various characters from students' animations. The oil paintings show Painter's simulation of oil on canvas. Figs. 17 and 19 are accompanied by supplemental videos.

Fig. 17 - Leo Oil Painting

Fig. 18 - Cowboy Oil Painting

Fig. 19 - Jaguar Oil Painting

Based on the criteria of visual appeal and painterly resemblance, the Leo painting is the most effective display of the tool. The form is easily distinguishable and the sharp shadows help to enunciate the form even further. The jaguar painting is more abstracted but still resembles a painting made by an artist. The cowboy painting is the most abstract of the three but still approximates a painting. Animations for Leo and the jaguar accompany the paper. Leo was modeled, textured and animated by Christine Li; the jaguar and cowboy were modeled, textured and animated by Krista Murphy.

CHAPTER IV

METHODOLOGY

The process for using Painter to achieve the illusion of brushstrokes in forms and figures for digital animation environments has been described briefly in this paper, and this section goes into detail about the concepts and equations used for the project. A more detailed look at which software was used as well as specific workflow appears in the next chapter.

In the simplest form, this project can be described in four steps:

1. 3D data was gathered from an animation to describe the scene.
2. Brush strokes were placed on the 3D surface.
3. Those strokes were projected into 2D and that data was saved as text files.
4. Those strokes were then rendered in Painter.

IV.1 Gathered 3D Object Data

The 3D information used for this project was in the form of Wavefront object files (OBJ files) that listed vertex positions, vertex normals, UV texture coordinates and a list of faces made up of vertices. A single file described an object in its basic form without additional information found in other formats; some examples of this information are texture maps, animation rigs and hierarchical transformations. Each vertex was listed with its world space location, or position in 3D space, and required no computation of transformations or deformations. Multiple objects could be exported to describe a

frame, and one OBJ file could not describe more than one object or more than one frame.

IV.2 Placed Brush Strokes

Once the OBJ files were exported, they were read in by the software and processed. Objects stored vertex and face information, and each object had access to vertex positions for each frame. The objects had an initialization frame that was used to propagate the brush strokes according to the parameters describing stroke position, number and orientation. Once this frame was processed, the objects were populated with particles, i.e. locations in space that contain information that is described later. The number of particles an object had was dependent on the mesh size and density of that object and the quantity of particles desired per face. Since the objects were defined by triangles, a particle's position, P, was defined by a 3D point describing the point's distance from each vertex in the face, P_1, P_2, P_3. This is referred to as barycentric coordinates (Warren et al. 1996) as shown below in Fig. 20.

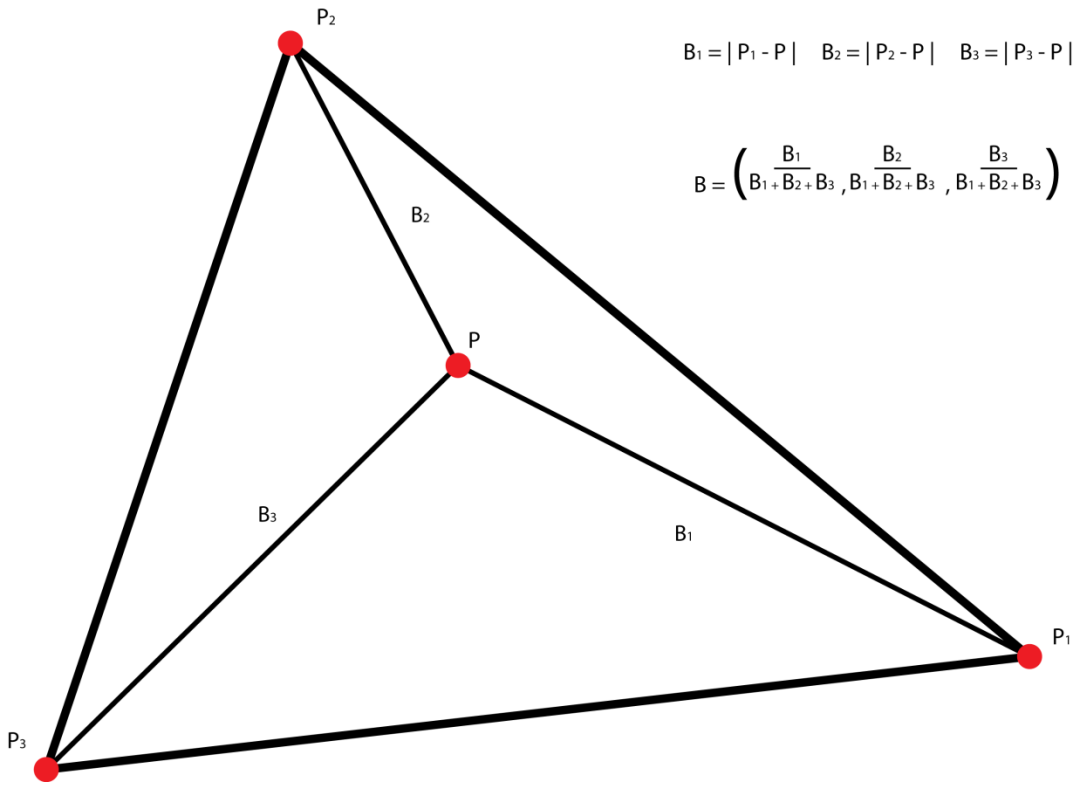

Fig. 20 – Barycentric Coordinates

Once the particle was positioned on the face, it needed to point in a direction the brush stroke would follow. This was done by supplying a "goal orientation", O, for all the strokes for all objects. This orientation vector was not guaranteed to lie on the plane of the face so some adjustment was required. The resulting vector, O`, was the intersecting line between the plane of the face and the plane containing O and N, the face normal. This was achieved by taking the cross product O and N, giving a vector that was perpendicular to both vectors, V, then taking the cross product of that vector and the face normal to obtain O`. The last piece of information stored was the endpoint of the stroke, P`, created by adding the orientation vector to the point of the particle. This is explained visually in Fig. 21 below.

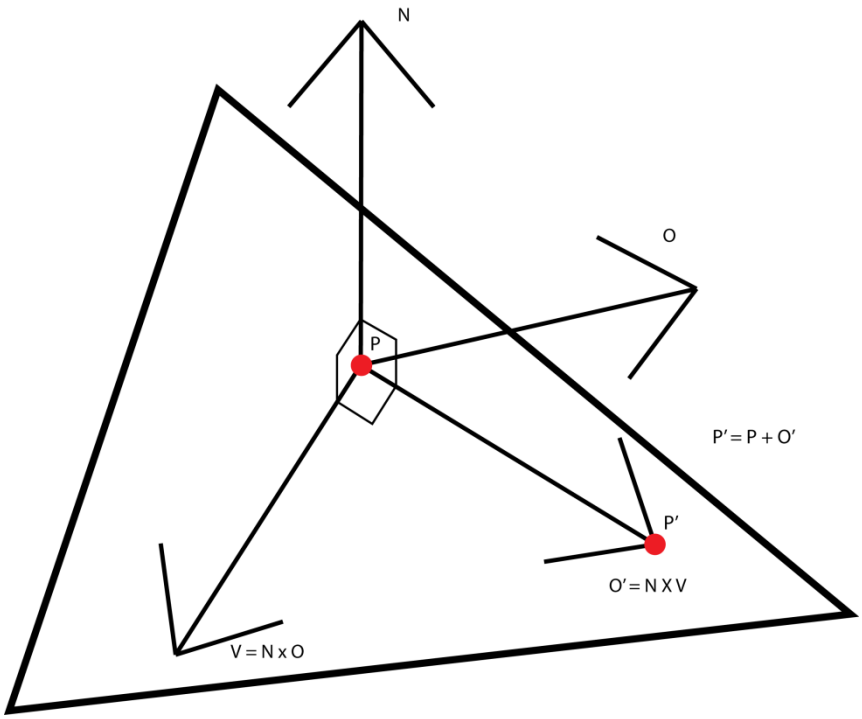

Fig. 21 – Orientation Diagram

The particles were also given a Red, Green, Blue (RGB) color between zero and two-hundred fifty five, a size that dictated the width of the brush used, and a length for the stroke that was stored as the magnitude of the orientation vector. These values were created by an artist for each object in the scene and given in the form of a mean value and a standard distribution to be stored in a text file. Values needed to be checked to make sure they do not fall outside a given range, a detailed description of which appears in chapter V.1.1.

IV.3 Camera Projection

The particles existed in a virtual 3D space but needed to be painted in a 2D paint package, so a virtual projection was required. To create the projection, a camera in virtual 3D space was created with the desired position and direction. A plane was placed in front of the camera to record where the particles needed to be drawn. A line was created between the particle and the camera with the plane in the middle of the two - if the plane was not between the camera and the particle, then the particle was considered "off camera" and did not appear on the screen. The point on the camera plane that intersected the line was the 2D location where the particle was drawn. Once the points were projected into 2D and saved, the artist submitted them to the engine for rendering.

IV.4 Corel® Rendering

As mentioned earlier, my renderer was designed to use Painter to handle the final stages of rendering. After creating paintings by hand in Painter, I was able to decide which paints I wanted to use and their corresponding media. To interface with Corel®, I created text files with instructions that tell the package where to paint, what brush to use, etc. A brief roadmap is given in this chapter to introduce the next chapter, which is a more in depth description of the code implementation.

CHAPTER V

IMPLEMENTATION

V.1 Creating Brush Strokes

V.1.1 Particle Creation

The process began with an artist creating an animation in an animation software package, using traditional rendering methods. I chose Maya (Beveridge 2012) due to my familiarity with the software. Most objects in 3D animations are stored as "quadrangulated meshes"- meaning that each face of the object has four corners, or vertices. For this process, each object had to be triangulated in order for the barycentric coordinates to work correctly. The artist then needed to export files containing object positions for each frame to be used by the software. I chose to use the OBJ file type, as it was relatively easy to read and write and also was compatible with multiple pieces of virtual 3D software. Each object in the scene needed unique definitions for the various attributes related to the final brushstrokes. These attributes included: brush type, brush size, number of brushstrokes per face, color and goal orientation. Given this information, each object was propagated with a specific number of particles. This number was a factor of the number of polygonal faces in the object, of the surface area of the entire object and a multiplier that artists used to control density of brushstrokes. Since these numbers were somewhat randomized, using a Gaussian distribution with a

mean and a standard deviation, the results sometimes needed to be adjusted and/or clamped. Gaussian, or normal, distribution is a random number distribution model that creates randomly generated numbers centered on a value, the mean, and can be sorted by their distance from that mean, the deviation. The Gaussian distribution method most frequently applied to brushstroke orientation. When using only random direction vectors in virtual 3D-space the resulting painting did not resemble the rendered model; rather it appeared to be several smears on a canvas, shown in Fig. 22. To address this particular problem, the direction vector was projected onto the plane of the surface by taking the cross product of the surface normal and the "goal orientation," then taking the cross product of the newly found vector and the surface normal. These cross products produced an orthonormal basis with a vector that lied on the surface of the object and lied in the same place as the "goal orientation" and surface normal. The other values were controlled by simply clamping them between some specified ranges to ensure the Gaussian distribution did not return a value outside the expected range.

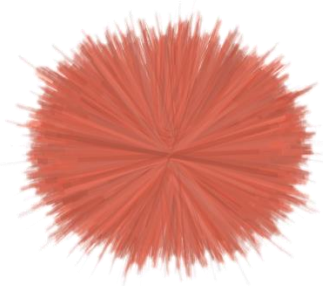

Fig. 22 - Improper Stroke Orientations

Once the objects were initialized with the starting values, they were displayed in a real-time OpenGL window that showed a preview painting vaguely resembling the final painted render. The preview did not handle physical phenomena such as paint blending or even multiple media – it was designed to give the artist an idea of what to expect from Painter. The reason for the facsimile was to allow the artist to quickly iterate through multiple versions of the piece until a satisfactory painting was achieved, similar to a rough preparatory sketch for a physical painting. Another benefit was that the artist can move through the virtual 3D painting and see it from multiple angles while working and not have to wait for a fully detailed render. Once satisfied, the artist then rendered the paintings in Painter.

V.1.2 Real-time Preview

In order to give artists control over the aesthetics of the animation, a real-time preview system had to be created. This preview mode provided instantaneous feedback for the artist and increased both productivity and quality by reducing the time spent waiting on results. The idea behind the real-time preview was to encourage a workflow similar to commercial animation software by making it easy to produce multiple iterations of ideas and images. Though the correlation between this preview and Corel® was not exact, it gave the artist a good idea of what to expect from the final render. Fig. 23 shows an example of what the artist saw in real-time when using the preview feature.

Fig. 23 - Real-time Preview

V.2 Using Painter

Painter allowed artists to record the placement and attributes of the paint brush while they painted and share that recording with other artists in the form of abbreviated text files that described every action the artist made. These actions told the software to start a brush stroke, where that stroke was placed, when to end the stroke as well as dictated brush type, size and color. Now the assignments of attributes to the 3D particles became necessary. As each particle was rendered, a snippet of text was created that described how that stroke looked.

V.2.1 Structured Recordings

The beginning of every recording file was a description of the entire scene. File name, date, canvas size and type, and random seed were all included in the file header. From

there, a sequential cycle of brush stroke descriptions began. A single stroke was defined in this way:

```
max_size_slider    1.29752
color red 160 green 106 blue 75
stroke_start
pnt x  329.295 y  282.88 time 100000 prs .71 tlt 0.08
pnt x  329.328 y  283.168 time 100000 prs .52 tlt 0.12
stroke_end
```

The "max_size_slider" was the brush size, followed by the RGB color on a 0-255 value scale. The next piece, "pnt," defined the x and y positions of the brush position. Time indicated when the stroke was made and was not needed for this project. Pressure was defined by "prs" and determined how much paint was applied to the canvas. Tilt, or "tlt", simulated a brush hitting the canvas at an angle. The stroke was closed by the "stroke_end" action. An average frame for the Leo animation had approximately 20,000 brush strokes.

V.2.2 Distributed Rendering

It was necessary to formulate some kind of distributed rendering system given that a single frame took anywhere between twenty minutes to two hours to render. Rendering also depended on the machine and the amount of memory leakage occurring. Memory

leaks occurred when Painter allocated memory to perform a task but then did not clear the memory after the task was completed. This allocation caused a pileup of stale memory that could not be reused and slowed down processes running on the machine or caused the software to crash. The distributed rendering system was achieved by using Google Drive, online software that connected folders on multiple machines as if they were on a network. The performance hit caused by internet transfer speeds was negligible since the paintings were only saved and transferred once the image was completely drawn. This approach made it easy to save straight from Painter to the Google Drive and share scripts and images without moving the files and their contents manually. The software also had an online webpage that allowed me to track progress of the renders from anywhere. This software ran on portable devices such as smart phones, allowing for remote monitoring.

CHAPTER VI

CONCLUSIONS

VI.1 Measuring Success

The goal of creating a temporally coherent animation by expanding upon *Painterly Rendering for Animation* (1996) and using Painter to simulate the application of paint to canvas was achieved successfully. Also, the rendered pieces proved that particles attached to a surface retain positional information between frames and maintain a flat, stylistic look. Although the animations produced by the software were up to the artistic standards of the industry, they took entirely too much time and hand-manipulation to be considered a perfect product. The long render times and hand manipulations were due mostly to the issue of using Painter for a task it was not designed to handle. The need for these manipulations raises the question of whether Painter was successful as a rendering engine.

Painter cannot be used as the rendering engine for a feature length animated movie. A known memory leak in the software and the lack of batch utilities makes this software ineffective for large-scale projects. However, if those two issues were fixed, this software would be a great solution for the problem of creating a painterly rendering system usable for feature length animated films. The ideal rendering engine would incorporate Painter's level of control and realism with the ideas of efficiency and distributed computing common in feature animation.

VI.2 Future Work

While I consider this project to be a success, there are three pieces I would suggest fixing in order to better this project. A self-contained real-time painterly rendering engine would make this project viable for use in the animation industry. This project needs its own paint simulator that can interface with the C++ code driving the real-time preview and make the experience more interactive for the artist. Also, to make the rendering process cost effective and practical for a feature length animation, a batch processing mode needs to exist that can run on a machine without user input. Last, a graphical plug-in for animation software like Autodesk Maya would keep the whole process contained to an area that the artist is already familiar with. These three items are out of scope for the project but address the three biggest hardships I faced while creating the animations. Future work should be done to incorporate stroke generation techniques and paint application simulations, then this software should run in real-time inside of commercial animation software packages.

VI.2.1 Paint Simulator

As mentioned earlier, Painter has an amazingly photorealistic paint simulation system that can create paintings that look almost real. This realism is what drew me to the package and is necessary for creating a painterly animation in this style. Future work should create a standalone paint simulator, or leverage existing simulation tools, and

connect it to the rest of the rendering engine. The paint simulations should be based on the work done by Lee et al. (1999) to simulate paint application to media in real-time because of the performance and speed introduce by the technique. Paint simulations done without Painter can be distributed across multiple machines, or processed in batch.

VI.2.2 Batch Mode

The frame times experienced during this animation are quick relative to studio standards. An hour spent rendering a frame is quick enough to get the rendering done because studios have area full of machines that only run processes for rendering. If this tool is to be used by those studios, it needs to be compatible with those machines without the need for a graphical interface or other interaction. Allowing for this option would be as easy as configuring the paint simulator to run without graphical interaction. This ability of batch processing would be best take advantage of if accessible from inside of commercial software such as Maya.

VI.2.3 Maya Plug-In

This last future idea is unrelated to the rendering itself but would greatly optimize workflow when using this rendering method. Running this tool inside Autodesk Maya would eliminate most of the overhead spent generating object description files and remove the discontinuity between the rendering and the rest of the animation process.

No other rendering method requires artists to leave the animation package to open another application for rendering. This discontinuity is a huge setback and one that needs to be remedied before mass use.

VI.3 Final Thoughts

A conceptual framework has been provided along with a software prototype in the hopes of bringing visibility to this stylistic technique. The paintings shown were created by me alone but I will distribute this software amongst colleagues to gain feedback and to give fellow artists access to the software. There is still work to be done but this project proves to be a positive starting point for the future.

REFERENCES

Baxter, Bill, Scheib, Vincent, Lin, Ming C., and Manocha, Dinesh. "DAB: interactive haptic painting with 3D virtual brushes." *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. New York: ACM, 2001. 461-68. Print.

Bénard, Pierre, Forrester Cole, Michael Kass, Igor Mordatch, James Hegarty, et. al. "Stylizing Animation by Example." *ACM Transactions on Graphics* 32.4 (2013): 1. Print.

Bénard, Pierre, Adrien Bousseau, and Joëlle Thollot. "State-of-the-Art Report on Temporal Coherence for Stylized Animations." *Computer Graphics Forum* 30.8 (2011): 2367-386. Print.

Beveridge, Crawford W. *Maya*. Vers. 2012. Mill Valley, CA: Autodesk, 2012. Computer software.

Cowpland, Michael. *Painter*. Vers. 2012. Ottawa, ON: Corel, 2012. Computer software.

Davis, Charles Barrett. Landscapes of Color. Diss. Texas A&M University, College Station, 2011.

Gooch, Bruch, Greg Coombe, and Peter Shirley. "Artistic Vision: Painterly Rendering Using Computer Vision Techniques." *Proceedings of the 2nd International Symposium on Non-photorealistic Animation and Rendering*. New York: ACM, 2002. 83-Ff. Print.

Haeberli, Paul. "Paint by Numbers: Abstract Image Representations." *Proceedings of the 17th Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM, 1990. 207-14. Print.

Hanrahan, Pat, David Salzman, and Larry Aupperle. "A Rapid Hierarchical Radiosity Algorithm." *Proceedings of the 18th Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM, 1991. 197-206. Print.

Hanrahan, Pat, and Wolfgang Krueger. "Reflection from Layered Surfaces Due to Subsurface Scattering." *Proceedings of the 20th Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM, 1993. 165-74. Print.

Hegde, Siddharth, Christos Gatzidis, and Feng Tian. "Painterly Rendering Techniques: A State-of-the-art Review of Current Approaches." *Computer Animation and Virtual Worlds* 24.1 (2013): 43-64. Print.

Hertzmann, Aaron. "Paint by Relaxation." *Computer Graphics International 2001. Proceedings*. Hong Kong: IEEE, 2001. 47-54. Print.

Kajiya, James T. "The Rendering Equation." *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM, 1986. 143-50. Print.

Kowalski, Michael A., Lee Markosian, J. D. Northrup, Lubomir Bourdev, Ronen Barzel, and Et Al. "Art-based Rendering of Fur, Grass, and Trees." *Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM, 1999. 433-438. Print.

Lee, J. "Simulating Oriental Black-ink Painting." *IEEE Computer Graphics and Applications* 19.3 (1999): 74-81. Print.

Maltin, Leonard, and Jerry Beck. *Of mice and magic: A history of American animated cartoons*. McGraw-Hill, 1980.

Markosian, Lee, Barbara J. Meier, Michael A. Kowalski, Loring S. Holden, J. D. Northrup, et al. "Art-based Rendering with Continuous Levels of Detail." *Proceedings of the 1st International Symposium on Non-photorealistic Animation and Rendering*. New York: ACM, 2000. 59-66. Print.

Meier, Barbara J. "Painterly Rendering for Animation." *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York: ACM, 1996. 477-84. Print.

Newell, M. E., R. G. Newell, and T. L. Sancha. "A Solution to the Hidden Surface Problem." *Proceedings of the ACM Annual Conference*. New York: ACM, 1972. 443-50. Print.

Noris, G., D. Sýkora, S. Coros, B. Whited, M. Simmons, and Et Al. "Temporal Noise Control for Sketchy Animation." *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*. New York: ACM, 2011. 93-98. Print.

Olsen, Sven C., Bruce A. Maxwell, and Bruce Gooch. "Interactive Vector Fields for Painterly Rendering." *Proceedings of Graphics Interface 2005*. Waterloo: Canadian Human-Computer Communications Society School of Computer Science, U of Waterloo, 2005. 241-47. Print.

Papari, Giuseppe, Nicolai Petkov, and Patrizio Campisi. "Artistic Edge and Corner Enhancing Smoothing." *IEEE Transactions on Image Processing* 16.10 (2007): 2449-462. Print.

Park, Youngsup, and Kyunghyun Yoon. "Painterly Animation Using Motion Maps." *Graphical Models* 70.1-2 (2008): 1-15. Print.

Strassmann, Steve. "Hairy Brushes." *ACM SIGGRAPH Computer Graphics* 20.4 (1986): 225-32. Print.

Vanderhaeghe, David, Pascal Barla, Joelle Thollot, and Francois X. Sillion. "Dynamic Point Distribution for Stroke-based Rendering." *Proceedings of the 18th Eurographics Conference on Rendering Techniques*. Aire-la-Ville: Eurographics Association, 2007. 139-46. Print.

Warren, Joe. "Barycentric Coordinates for Convex Polytopes." *Advances in Computational Mathematics* 6.1 (1996): 97-108. Print.

Watson, Andrew B., and Jr. Albert J. Ahumada. "Model of Human Visual-motion Sensing." *Journal of the Optical Society of America A* 2.2 (1985): 322-41. Print.

Zeng, Kun, Mingtian Zhao, Caiming Xiong, and Song-Chun Zhu. "From Image Parsing to Painterly Rendering." *ACM Transactions on Graphics* 29.1 (2009): 1-11. Print.

Zwicker, Matthias, Mark Pauly, Oliver Knoll, and Markus Gross. "Pointshop 3D: An Interactive System for Point-based Surface Editing." *ACM Transactions on Graphics* 21.3 (2002): 322-29. Print.