Real Time Sumi-E Style Rendering Techniques

Santiago Gallo Beruben

A Dissertation

Presented to the University of Dublin, Trinity College

in partial fulfilment of the requirements for the degree of

Master of Science in Computer Science (Augmented and Virtual Reality)

Supervisor: John Dingliana

September 7, 2020

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Santiago Gallo Beruben

September 7, 2020

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

____

Santiago Gallo Beruben

September 7, 2020

Acknowledgments

I would like to start off by thanking my professors, particularly my dissertation advisor, Dr. John Dingliana for his guidance through this project.
I would also like to thank my friends from from across the world, the people who pushed me to take a leap of faith and enroll on this program, the people who gave me all of their unconditional support in a multitude of ways and helped me get through this tough year.
Last but not least I would like to thank my loving family as I would not be writing this were it not for all of their support.

Santiago Gallo Beruben

University of Dublin, Trinity College
September 2020

Real Time Sumi-E Style Rendering Techniques

Santiago Gallo Beruben, Master of Science in Computer Science
University of Dublin, Trinity College, 2020

Supervisor: John Dingliana

Computer generated imagery (CGI) is commonplace in media such as video games and cinema, both areas that are at the forefront of developments in the field of computer graphics. Advancements in the field have had for the most part focused on trying to achieve realism. However, there are situations in which realism is not desirable as information can be abstracted and conveyed in a different yet more efficient manner, leading to the need for non-photorealistic rendering (NPR).

A style of NPR which has not had as much widespread adoption beyond the PlayStation 2 video game Ōkami is that of sumi-e or ink wash rendering; referring to a set of brush painting techniques used in East Asian countries, particularly Japan, China and Korea. Sumi-e makes use of black ink in a white paper canvas to create simple drawings via single, well defined strokes. Sumi-e rendering techniques, like other areas of NPR focus on edge extraction and drawing of strokes and edges, drawing of textures and interior shading but have a different approach so that the desired art style can be achieved.

The goal of this project is to research and implement previously established techniques used for sumi-e style rendering and develop a sumi-e rendering pipeline. The rendering pipeline will be applied to various 3D models and will be used to verify whether sumi-e rendering is suitable for media such as video games.

Contents

Acknowledgments **iii**

Abstract **iv**

List of Tables **vii**

List of Figures **viii**

Chapter 1 Introduction **1**

1.1 Motivation 2

1.2 Objective 3

1.3 Assumptions 3

1.4 Limitations 3

Chapter 2 Background Research **5**

2.1 Non-Photorealistic Rendering 5

2.2 Painting in NPR 6

2.3 Sumi-E Painting 6

2.4 Previous Work 8

2.4.1 State of the Art 11

Chapter 3 Design and Implementation **17**

3.1 Comparison of Rendered Images and Sumi-e Paintings 17

3.2 Rendering Pipeline Design 18

3.3 Final Implementation 28

Chapter 4 Results **31**
4.1 Tests and Comparison with Sumi-E 31
4.2 Performance 37
4.3 Conclusions 38

Chapter 5 Future Work **42**

Bibliography **43**

Appendices **44**

List of Tables

2.1 Set of green color values used by a cel shader as an example of how colors are picked based on light intensity values. Displays color names, RGB values from 0-255 and percentages used by a GLSL shader, results shown in figure 2.8 14

3.1 Table of color values used by the cel shader implemented in GLSL shader. Results shown in figure 3.8 24

4.1 Table displaying performance for each experiment run when testing the sumi-e rendering pipeline. Tests were done rendering images at a resolution of 1024 x 768 40

List of Figures

1.1 Screenshot of the 2005 PlayStation2 video game, Ōkami. Copyright Capcom Co., Ltd 2

2.1 A painting made using the sumi-e style, plate XVI from 'On the laws of Japanese Painting' by Henry P. Bowie [1] 8
2.2 Bamboo, Sparrow and Rain by Torei Nishigawa. Plate VII from 'On the laws of Japanese Painting' by Henry P. Bowie [1] 9
2.3 Shrimp and leaf drawing generated by Strassman's technique discussed in 'Hairy Brushes' by Strassman [2] 10
2.4 Different brush strokes generated using Binh Pham's technique discussed in 'Expressive Brushtrokes' [3] 10
2.5 Structure of the system used by Shin-Jin Kang et al in 'Hardware-accelerated real-time rendering for 3d sumi-e painting'[4] 11
2.6 Example of how texture mapping gets closer to the sumi-e look. (a) shows a 3D mesh of a cow with its silhouette extracted and drawn in black. (b) shows the same 3D mesh using silhouette extraction and sphere mapping used in conjunction with a brush texture for drawing of edges. Taken from Shin-Jin Kang et al's work 'Hardware-accelerated real-time rendering for 3d sumi-e painting' [4] 13
2.7 Brush stroke textures used when drawing edges while making use of sphere mapping, as per the technique shown in both 'Hardware-accelerated real-time rendering for 3d sumi-e painting'[4] as well as 'Hardware-Accelerated Sumi-e Painting for 3D Objects'[5] 13

2.8 Look up table with different colors based on different light intensity values. The lower the light intensity, the darker the color and vice versa. Based on documentation by Hutchins and Kim [6] and colors shown in table 2.1 14
2.9 Tone texture used for interior shading [5] 15
2.10 Output from using a tone texture for interior shading on a sphere [4] . 15
2.11 Output of the rendering pipeline making use of edge detection, interior shading via tone texture and paper effect from [4] 16

3.1 Rendering pipeline used for this project, based on similar previous works in conjunction with other NPR techniques 19
3.2 Edge detection by using equation 2.1. From left to right, ϵ threshold value is set to 0.2, 0.5, 0.7 and 1.0. As ϵ increases, edges grow thicker, to the point of darkening the entire rendered image at a maximum of 1.0 20
3.3 Stanford Bunny mesh rendered with different lit and unlit outline edge values. Detected edges are drawn in black whereas interior of the mesh is drawn in white 20
3.4 Drawing of edges with stroke effect using different textures on the Stanford Bunny 3D mesh 21
3.5 Drawing of edges using wobble distortion on the same stroke texture. Edge drawing can be altered slightly based on the stroke texture 22
3.6 Drawing of edges using solid colors in gray scale, blending in as the colors get lighter to the point that edges are not seen halfway through and interior edges blend in with the interior as edges are drawn with at their brightest color. 22
3.7 Tone texture shading based on the work by Shin-Jin Kang et al [4]. The usage of the tone texture instead of a cel shader allows for a bleeding effect of ink in paper. Left image shows the initial implementation taken from previous work and the right image shows a modified version in which sampled values are clamped. Older implementation code shown in appendix B, list 3 code shown in appendix B, list 4 23
3.8 Cel-shader used to pick black and white colors based on light intensity on the Stanford Bunny 3D mesh. Code shown in appendix B, list 4 . . 23

3.9 Comparison of how a cel-shader and a tone texture shader while making use of interior textures. Code shown in appendix B, list 5 24
3.10 Texture used for generating the skybox and achieving a paper effect . . 25
3.11 Skybox drawn based on the texture shown in fig 3.10 with a drawn mesh of the Stanford bunny 26
3.12 Stanford bunny rendered without transparency effects by changing the alpha channel as well as two examples with alpha channel transparency. Middle image makes use of back face culling whereas the right image does not and displays data taken from the back face of the 3D mesh . . 27
3.13 Stanford bunny rendered without changes to the average colors alpha threshold as well as the alpha threshold skip values. Code shown in 7, appendix B 28
3.14 User interface elements added with AntTweakbar for manipulating behavior of the rendering pipeline 30

4.1 Rendering pipeline output using the Stanford bunny on different positions with variation in outline thickness and usage of cel-shader or tone texture shader 32
4.2 Rendered images of the Stanford bunny using badly picked values for edges drawn with stroke textures using sphere mapping 32
4.3 Closeup of the Stanford Bunny 3D mesh when using alpha transparency effects. Shows back face culling inconsistencies and how slight variations in parameters can help diminish the issue when present 34
4.4 Stanford Bunny 3D mesh drawn in both the background and foregound, without and with face culling enabled 34
4.5 Rendered image of a bamboo tree drawn using the cel-shader from the rendering pipeline. Left side shows no alpha transparency effects added. Right side shows alpha transparency effects 35
4.6 Closeup rendered images of the bamboo tree. Top section makes use of the tone-texture shader and bottom texture makes use of the cel-shader. Left section makes no use of textures for interior shading and right section makes use of textures for interior shading 36

4.7 Closeup to images of bamboo leaves. Left image is taken from 2.2 whereas center and right image are taken from 4.6. Center image makes use of tone texture shading whereas the right image makes use of the cel-shader 36
4.8 Orange texture taken from [7]. 37
4.9 Sphere mesh generated using Blender. Top section makes use of the cel-shader whereas bottom section makes use of the tone-texture shader. Left section makes no use of transparency/alpha effects, middle section makes use of transparency/alpha effects and right section makes use of transparency/alpha effects in conjunction with an orange peel texture taken from [7] 38
4.10 Similar sphere to that of figure 4.9, using a higher vertex count mesh made using Blender 39
4.11 Low-poly airplane 3D model from [8] rendered with a cel-shader (left) and a tone-texture shader (right) with transparency effects disabled (top) and enabled(bottom) 40
4.12 Modified low-poly airplane 3D model from [8], edited in Blender to have a much higher vertex count. Rendered with a cel-shader (left) and a tone-texture shader (right) with transparency effects disabled (top) and enabled(bottom) 41

Chapter 1

Introduction

This dissertation consists on the research, implementation and improvement of non-photorealistic rendering (NPR) techniques which focus on rendering images based on sumi-e paintings. Algorithms researched and developed for this dissertation project focus on sumi-e painting aspects applied to three points of interest for NPR techniques.

- Drawing of strokes and edges
- Drawing of textures
- Interior shading

The implementation of the algorithms researched and developed for this dissertation are all running in C++ as well as the Open Graphics Language (OpenGL) using the OpenGL Shading Language (GLSL) on a Windows 10 PC using Visual Studio 2019 Community Edition running on an Intel 8700K processor and an Nvidia GTX 1080 graphics card.

Through this chapter, motivations and objectives for this dissertation will be discussed. Chapter 2 focuses on background research of NPR and sumi-e painting, as well as the state of the art for sumi-e rendering techniques. Chapter 3 focuses on the design implementation of a rendering pipeline which makes use of techniques researched. Chapter 4 discusses the resulting images from the rendering pipeline, comparing different approaches taken while using sumi-e rendering and discusses whether techniques were successful or not at matching the characteristics of sumi-e. Chapter 5 discusses future work and areas of improvement for the end product.

1.1 Motivation

The field of computer graphics has had major focus on realism, so much in fact that it has been one of the major driving forces behind its research and development, leading to investments worth millions of dollars[9]. However, there are situations in which achieving realism is disadvantageous to the intent of a computer graphics application. This leads to non photo realistic rendering (NPR), an area of computer graphics which does not focus on achieving realism and instead focuses on replicating the look and feel of art styles used in media such as cartoons, technical drawings, panting and pencil drawings.

Figure 1.1: Screenshot of the 2005 PlayStation2 video game, Ōkami. Copyright Capcom Co., Ltd

Despite being received with high praise and critical acclaim from fans and press alike, as well as being ported to a multitude of platforms after its original release, few games if any have followed in Ōkami's footsteps.While the game does an admirable job at displaying the sumi-e style in a video game; especially for what was originally a PlayStation2 video game, there are definite areas of improvement as it still looks very much like a video game constrained by its hardware. 3D models look like 3D models with a sumi-e look and feel instead of appearing to be parts of a sumi-e painting drawn in a canvas. Only elements of background scenery such as trees and mountains in the background blend in as part of a painting. This is an area in which Ōkami's accomplishments in rendering can be greatly improved upon.

1.2 Objective

The objective of this project is to conduct research on sumi-e focused, real- time rendering techniques. Algorithms that are implemented will be used to design and implement a new rendering pipeline which adheres to principles of sumi-e painting. Once finished, the rendering pipeline will compared with previous works and sumi-e paintings while also evaluating whether sumi-e real-time rendering techniques could be of use in media such as video games.

The final rendering pipeline will be used to create images consisting of multiple elements such as foreground and background 3D meshes while also animating other elements. Performance will also be of interest due to the real time and interactive nature of video games as a medium. Algorithms researched and the final rendering pipeline will be implemented in C++ using OpenGL and GLSL for shaders.

1.3 Assumptions

- Techniques researched and implemented will work repeatedly, regardless of the meshes used
- The project will verify wether sumi-e real-time rendering techniques are feasible in media such as video games
- Performance of the final rendering pipeline will be measured to verify its likelihood to be used in video games
- The rendering pipeline will make use of vertices and normals for all 3D models used
- Model texture coordinates will be an optional parameter when rendering 3D meshes which make use of textures

1.4 Limitations

- Implementation will not work with graphics libraries beyond OpenGL

- The project will be implemented in OpenGL using C++ and Visual Studio 2019, Community Edition in order enable its rendering pipeline
- The project will only make use of colors in the gray scale as sumi-e painting is limited to said color palette
- The project will use a pre-existing approach for detecting of and drawing edges and strokes as a baseline
- The project will use a pre-existing approach for using textures as a baseline
- The project will use pre-existing approaches for Interior shading as a baseline

Chapter 2

Background Research

2.1 Non-Photorealistic Rendering

In the field of computer graphics, achieving photo realism has been a major driving force behind research and development. Media productions such as film and video games convey realism through CGI by making use of multiple advancements from the field of computer graphics. However, photo realism is of little use in cases where high detail can hinder the transmission of information and ideas when using CGI.

In some cases the removal of detail makes it is easier to send information and convey ideas, as described by Gooch & Gooch;

"A sailboat builder would certainly prefer technical drawings of blueprints, while someone who simply wanted to communicate the idea of a sailboat may only need to draw a shape representing the boat and a triangle representing the sail."[10]

It is due to this necessity that the field of NPR exists. By making use of NPR techniques, computer graphics can remove detail from an image while also communicating more efficiently the information and ideas said images intend to transmit. This due to the fact that NPR techniques, as the artistic styles they emulate; focus on displaying the essential parts of the objects drawn/rendered. Having a major focus on stylization of an art style and communication of ideas, NPR makes equal use of artistic techniques and scientific research to develop algorithms which simulate the usage of

different artistic tools and instruments.

Different techniques of NPR focus on matching the use of various artistic instruments, in order to match the visual style of different media whether it is drawings, cartoons or painting. They can also focus on different media such as pencil drawings or painting. This project will focus on painting as that is the media type of sumi-e.

2.2 Painting in NPR

In order to match characteristics of an art style, NPR techniques need to replicate the following three components of a painting/drawing as mentioned by Gooch & Gooch [10]:

- Applicator - the brush used
- Medium - oil paint, watercolors, sumi
- Substrate - the canvas or paper used

Sumi-e has different characteristics for its applicator, medium and substrate compared to other artistic techniques, there are even slight difference between sumi-e and similar styles from China and Korea. The characteristics of paintings can be easily associated with major elements of NPR techniques that this research project focuses on:

- Drawing of strokes and edges - applicator and medium
- Drawing of textures - medium and substrate
- Interior shading - medium and substrate

2.3 Sumi-E Painting

Sumi-E, also known as ink wash painting is an art style developed in East Asia; particularly Japan, China and Korea. It focuses on using black ink similar to that used in Chinese and Japanese calligraphy, which preceded this style of painting in China and

Japan. As characters became more complex different writing techniques developed, painting techniques which made use of the same instruments were created [1].

In sumi-e, sumi refers to the 'ink' which is the distinguishing feature of Japanese painting of this style compared to its peers. Similar to how Japanese calligraphy works, the only color employed when using sumi is black, as in order to fabricate sumi, soot obtained from burning plants is necessary. Soot used for fabricating sumi comes from burning plants such as *juncus communis*, bull rush, or *thesessamen orientalis.* These materials are then turned into a 'cake' which must be moisturized when in use and rubbed into a stone in order to create 'ink' or sumi [1].

Besides the usage of sumi, this art style uses a special type of canvas; a unique kind of paper called *torinoko* which translates literally to "hen's egg" paper. Torinoko gets its namesake from its color and texture which is similar to the appearance of a hen's egg. Despite its name and likeness, it is actually made from a shrub called gampi. The paper's texture allows for the easy flow of brush strokes and due to this, torinoko has been used not only in sumi-e but also wood block paintings and calligraphy. Torinoko is also known for being longer lasting and more resilient than its alternatives such as silk canvases used in Chinese ink wash paintings [1].

The sumi-e technique is distinguished from others not only by the materials used by also by the way in which paintings are made. Outlines are seldom made before painting, meaning that each stroke must be thought of fully before being drawn in order to properly fulfill its intent when creating a painting. This is another influence of calligraphy over sumi-e [1]. Strokes must fulfill a specific role set by the painter reflecting before performing each stroke, another influence of calligraphy on sumi-e [1]. This

Figures 2.1 and 2.2 display examples of a sumi-e pantings taken from 'On the Laws of Japanese Painting' by Henry P. Bowie [1]. As can be seen, sumi-e paintings are made out of long, broad strokes in grayscale, using a white canvas and display elements of nature such as a flora, fauna and scenery such as birds, tres and elements of the natural landscape, all common subjects in sumi-e paintings.

Figure 2.1: A painting made using the sumi-e style, plate XVI from 'On the laws of Japanese Painting' by Henry P. Bowie [1]

2.4 Previous Work

There have been a multitude of systems used for sumi-e rendering such as the work done by Steven Strassman [2] which was done in order to simulate how brushes would produce strokes onto paper by generating lists of positions and pressure points via keyboard input, shown in Figure 2.3.

Another work in sumi-e related techniques is that of Expressive Brushes by Binh Pham [3]. In this work, Pham represented strokes via three components: trajectory, thickness and shade. Trajectory is modeled via a cubic B-Spline, stroke thickness is modeled by a set of points along the trajectory's curve and shade is determined along each trajectory. Figure 2.4 displays different e strokes generated using Pham's algorithm.

Figure 2.2: Bamboo, Sparrow and Rain by Torei Nishigawa. Plate VII from 'On the laws of Japanese Painting' by Henry P. Bowie [1]

While these early generate images which match characteristics of sumi-e, they are not meant for use in media that makes heavy use of 3D model, as is the case of video games and certain animated works. For research to be useful on interactive entertainment media and other kinds of animated works, techniques must make use of 3D polygonal meshes. Techniques that focus on video games must also put emphasis on real-time performance due to their interactive nature as a medium; as performance and responsiveness are of higher importance than when doing offline rendering. One of the few examples of sumi-e being used in video games is the PlayStatoin2 video game "Ōkami"[11], shown in Figure 1.1.

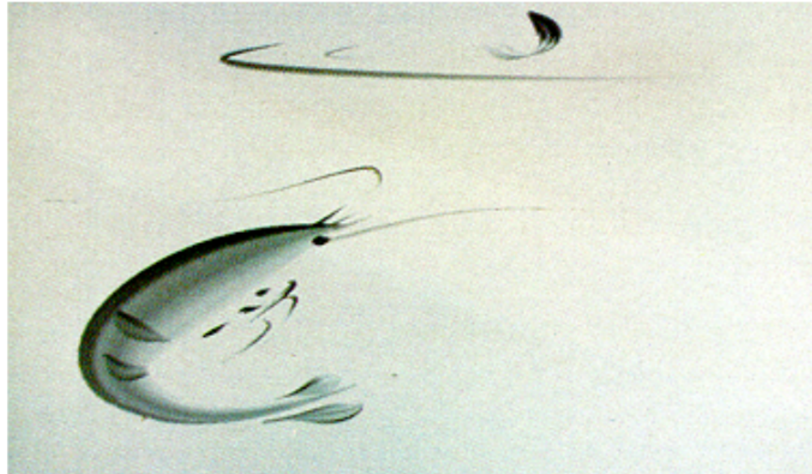

Figure 2.3: Shrimp and leaf drawing generated by Strassman's technique discussed in 'Hairy Brushes' by Strassman [2]

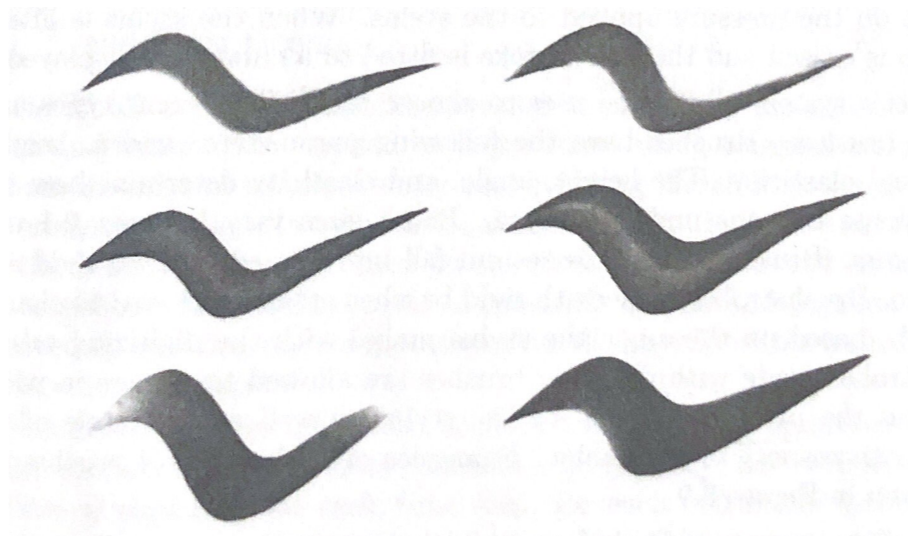

Figure 2.4: Different brush strokes generated using Binh Pham's technique discussed in 'Expressive Brushtrokes' [3]

The following techniques are all meant to be run in real time and make use of 3D meshes to match the look of sumi-e paintings by focusing on various elements used by NPR techniques; drawing of strokes and edges, drawing of textures and Interior shading. They all focus on replicating the usage of elements mentioned by Bruce Gooch; applicator, medium and substrate which correspond to a paintbrush, sumi ink

and torinoko paper.

2.4.1 State of the Art

One of the works by Shin-Jin Kang et al [4] demonstrates a rendering pipeline used to render 3D objets in the sumi-e art style. This project is implemented in shaders in order to make use of the graphics processing unit (GPU). The algorithm is split intro three major components: silhouette outlining, interior shading and paper effect which are shown in figure 2.5. Each of the major components on the system make use of a specific input in order to achieve the sumi-e look. Another work by Park et al[5] goes over similar techniques which focus more on a Korean interpretation of sumi-e but also covers effects that apply to any form of ink wash painting such as sumi-e; brush stroke effect/silhouette outlining.

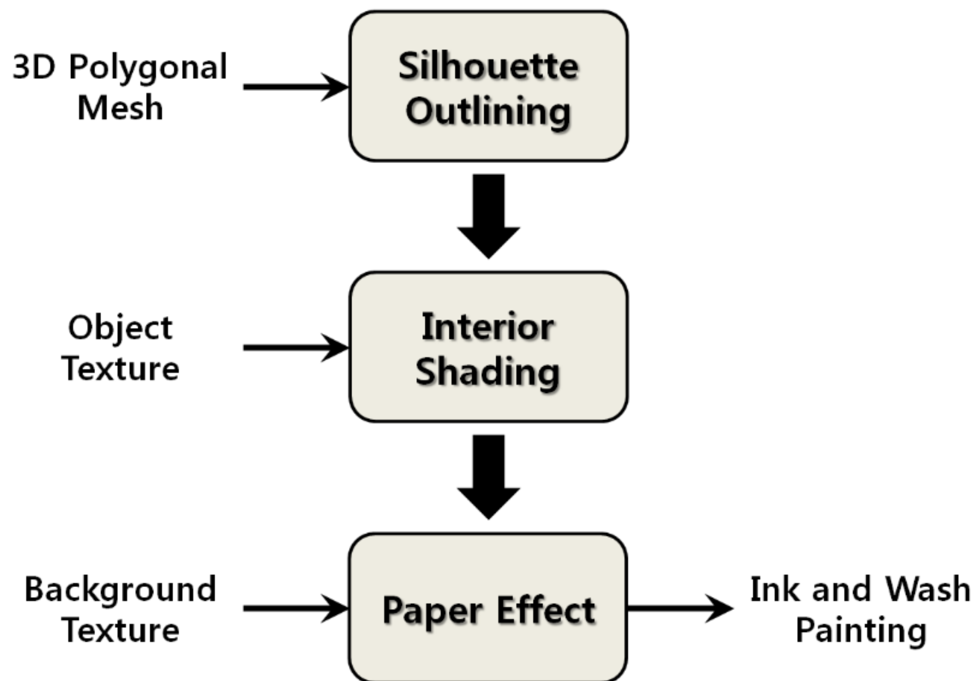

Figure 2.5: Structure of the system used by Shin-Jin Kang et al in 'Hardware-accelerated real-time rendering for 3d sumi-e painting'[4]

In order to achieve a sumi-e look and feel, it is necessary to outline the silhouettes

of objects or rather, draw the edges and add a brush stroke effect to edges. Detecting edges can be achieved by taking into account that a 3D model's vertices v can be determined to be edges if their normal vector N_v are perpendicular to their view vector V. Should the dot product of N_v and V sit between 0 and ϵ (threshold value for edge detection), then the vertex is to be considered an edge. This can be shown in Equation 2.1. The bigger the threshold value is, the thicker the edges can be and should an edge be detected, we can set its color to black, else it will be set to white [4].

$$0 \leq N_v \cdot V \leq \epsilon \tag{2.1}$$

Once edges are detected, they can more closely match sumi-e by use of textures. To do so, textures are used in conjunction with sphere mapping. We take the vector from the vertex to the camera v, normalized to $\hat{v}$. Vertex normal n is then to be transformed into view coordinates, $\hat{v}$. Afterwards a reflected vector $r(r_x, r_y, r_z)$ is obtained in equation 2.2, from which equation 2.3 and 2.4 are used to obtain texture coordinates (t_u, t_v).

$$r = 2(\hat{n} \cdot \hat{v})\hat{n} - \hat{v} \tag{2.2}$$

$$m = 2\sqrt{{r_x}^2 + {r_y}^2 + (r_z + 1)^2} \tag{2.3}$$

$$t_u = \frac{r_x}{m} + \frac{1}{2}, t_v = \frac{r_y}{m} + \frac{1}{2} \tag{2.4}$$

Once texture coordinates are obtained, brush stroke textures are used instead of black, as shown in figure 2.6. In another work by Shin-Jin Kang and others [5], the same technique for drawing of edges while making use of brush stroke textures with sphere mapping is shown. Figure 2.7 displays textures taken from both works [4], [5] for drawing edges.

Kang et al's approach to interior shading is similar to that of cel-shading but with the addition of textures instead of using look-up tables. Cel-shading is a technique used to emulate the look of traditional cel animation, one approach to achieve cel-shading is by measuring light intensity and color areas based on its value. In order to do so, at each vertex, the Lambertian is obtained at each vertex by taking the normal $\vec{n}$, the light direction $\vec{l}$ and calculating the dot product between them as shown in equation 2.5 [6].

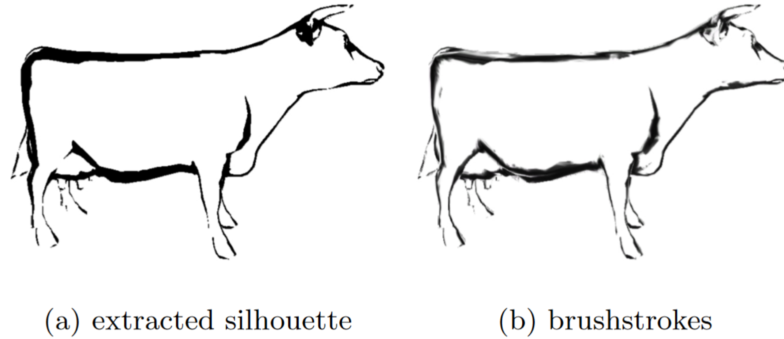

Figure 2.6: Example of how texture mapping gets closer to the sumi-e look. (a) shows a 3D mesh of a cow with its silhouette extracted and drawn in black. (b) shows the same 3D mesh using silhouette extraction and sphere mapping used in conjunction with a brush texture for drawing of edges. Taken from Shin-Jin Kang et al's work 'Hardware-accelerated real-time rendering for 3d sumi-e painting' [4]

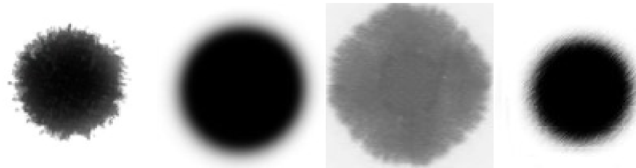

Figure 2.7: Brush stroke textures used when drawing edges while making use of sphere mapping, as per the technique shown in both 'Hardware-accelerated real-time rendering for 3d sumi-e painting'[4] as well as 'Hardware-Accelerated Sumi-e Painting for 3D Objects'[5]

$$-1 \leq \vec{n} \cdot \vec{l} \leq 1 \tag{2.5}$$

Once light intensity is obtained, colors can be assigned to a mesh based on its values. As per the work by Hutchins and Kim, anything between below 0 should be set to the darkest tone/color whereas between 0 and 1 can be assigned to previously specified colors, as for anything above 1 it is to be set to the brighest tone/color available [6]. These colors can be arranged on a look up table, using different light intensity values to determine colors. Figure 2.8 displays a lookup table of colors based on their RGB values and light intensity, as well as a resulting image of the Stanford bunny. This was implemented in C++/OpenGL.

Color	RGB Value	RGB Value %
Green Yellow	(173, 255, 47)	(0.678, 1.0, 0.184)
Lime Green - Dark Gray 1	(50, 205, 50)	(0.196, 0.804, 0.196)
Forest Green	(34, 139, 34)	(0.133, 0.545, 0.133)
Green	(0, 128, 0)	(0.0, 0.502, 0.0)

Table 2.1: Set of green color values used by a cel shader as an example of how colors are picked based on light intensity values. Displays color names, RGB values from 0-255 and percentages used by a GLSL shader, results shown in figure 2.8

Figure 2.8: Look up table with different colors based on different light intensity values. The lower the light intensity, the darker the color and vice versa. Based on documentation by Hutchins and Kim [6] and colors shown in table 2.1

Shin-Jin Kang et al's approach while similar to cel-shading, differs slightly in how colors are mapped. In this work, a tone texture is used in order to obtain the color of each vertex, shown in figure 2.9. In order to make transitioning from one color shade to another smooth, Gaussian blur is applied to the tone texture. [4].

Similar to a cel-shader, light intensity is calculated at each vertex, determining how much light lands at each pixel s. This is shown in equation 2.6 which is the same operation shown in equation 2.5. Similar the approach of a cel shader lookup table, light intensity values are clamped between 0 and 1 as shown in equation 2.7.

$$s = \vec{n} \cdot \vec{l} \tag{2.6}$$

$$s = \in [0, 1] \tag{2.7}$$

Figure 2.9: Tone texture used for interior shading [5]

Light intensity s is then used to scale diffuse color of an object based on the sampled colors taken from a texture. Afterwards, luminance is obtained from the sampled texture value in order to obtain u texture coordinates from the tone texture, as shown in equation 2.9. As the relevant values of the tone texture are at the X axis, tone texture coordinate v is a static value at 0.5 as the tone texture works in a similar fashion to a cel shader lookup table but using more calculations beyond obtaining light intensity and assigning a color based on its value.

$$diffuseColor = s(r, g, b, a) \tag{2.8}$$

$$u = min(s(0.3r + 0.59g + 0.11b), 1), v = 0.5 \tag{2.9}$$

Once texture coordinates are fetched, color values can be mapped to the 3D mesh giving interior shading colors based on the tone texture, as shown in figure 2.10.

Figure 2.10: Output from using a tone texture for interior shading on a sphere [4]

One other element of sumi-e is that corresponding to the canvas used, in this case torinoko paper. There is a need to create effects that simulate the usage of paper as a canvas. The work by Shin-Jin Kang et al in 'Hardware-accelerated real-time rendering

for 3d sumi-e painting'[4] generates an ink dispersion effectin conjunction with a paper effect, shown in figure 2.11.

Figure 2.11: Output of the rendering pipeline making use of edge detection, interior shading via tone texture and paper effect from [4]

One approach to having a paper effect as well as ink dispersion is by making use of the alpha channel. Both the works by [5] and [12] make use of the alpha channel for specific parts of their rendering pipeline.

The Nong-Dam effect can be used to convey variations in luminosity as ink tones change from dark to bright, shown in equation 2.10 where V_a refers to the alpha channel value of the Nong-Dam effect, V refers to the vertex position in view space, C refers to the Nong-Dam Effect center position and N refers to the view space vertex normal.

$$V_a = (V - C) \cdot N \tag{2.10}$$

Chapter 3

Design and Implementation

This chapter covers design and implementation details on the rendering pipeline implemented.

3.1 Comparison of Rendered Images and Sumi-e Paintings

In order to render 3D images that target sumi-e as their art style, a rendering pipeline must adhere to main points of sumi-e paintings discussed in 'On the laws of japanese painting' [1], such as:

- Usage of only black and white colors
- Usage of ink (sumi) as a medium and egg shell paper (torinoko) as a canvas with a brush as an applicator
- Usage of long/broad brush strokes
- Lack of sketching/drawing of outlines done before usage of the paint brush

Usage of black and white relies on defining colors by either making use of a lookup table or a tone texture. Replicating the use of ink, eggshell paper and a paintbrush requires more specific techniques. To do so, the rendering techniques used for this rendering pipeline make use of textures for both simulating the use of a paintbrush

when drawing edges as well as eggshell paper as a canvas/substrate. Textures may also be used when drawing the interior of a 3D mesh if necessary.

Drawing of edges are done by first doing edge detection followed by the usage of textures to match the usage of long/broad brush strokes used to draw images. One aspect of sumi-e painting which is easy to achieve due how the rendering pipeline works is the lack of previous sketching before the use of a paintbrush.

Sumi-e focuses on conveying images of nature; bamboo, birds and flowers as well as other different elements of flora and fauna being major subjects. Paintings such as those shown in figures 2.1 and figure 2.2 from 'On the laws of Japanese Painting'[1]. Still, the rendering pipeline can be applied to any 3D mesh in order to attempt matching sumi-e styled images.

3.2 Rendering Pipeline Design

The rendering pipeline designed for this project is based on multiple techniques, some used by previous sumi-e research projects and others from other NPR rendering techniques. One example used as a starting point is that by Kang et al in "Hardware-accelerated real-time rendering for 3d sumi-e painting" [4]. The rendering implemented consists of multiple steps as shown in figure 2.5; edge detection, drawing of edges with lit and unlit outline thickness and stroke textures or solid colors, wobble distortion of outline edges when using stroke textures, tone texture or cel-shader for interior shading in addition to optional use of 3D model textures being used for interior shading as well as creation of a paper effect via alpha channel values with varying degrees of transparency set by an average color threshold value used for setting alpha channel values as well as an alpha channel threshold value used for discarding pixels. A diagram detailing the flow of the rendering pipeline is shown in figure 3.1.

Edge Detection

Edge detection is done similar to how it was developed in previous works as shown in equation 2.1. The bigger the threshold value, the thicker the edges, as shown in figure 3.2. A 3D mesh of the Stanford Bunny is taken from the Stanford 3D Scanning Repository[13] and converted to an obj file using Blender in order to make it compat-

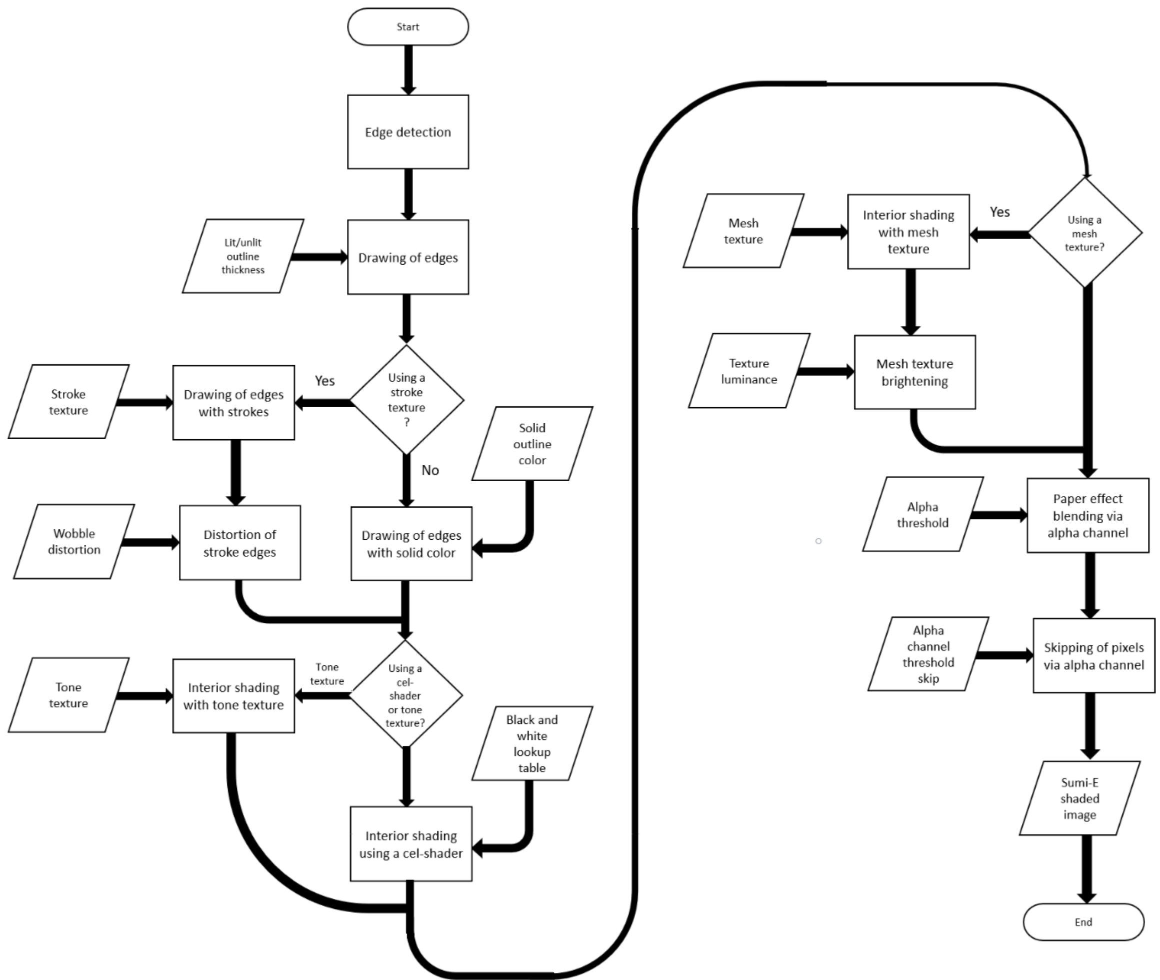

Figure 3.1: Rendering pipeline used for this project, based on similar previous works in conjunction with other NPR techniques

ible with the model loading libraries developed for this rendering pipeline. It is then rendered with edges drawn in black and its interior is drawn in white to better show edges. A small segment of the shader used to achieve this effect is shown in appendix B, listing 1.

While the ϵ threshold value can be determined manually, parameters for edge detection can be changed slightly, by making use of lit and unlit outline thickness values to draw edges in a more natural way, allowing the edges of lit/brighter areas to have different thickness to that of unlit areas and vice versa, covered in [14]. This can be

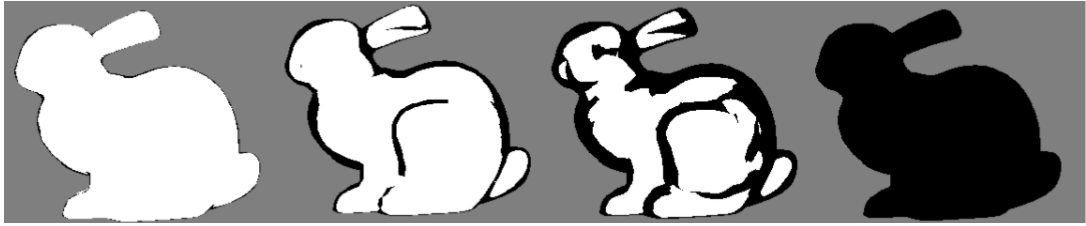

Figure 3.2: Edge detection by using equation 2.1. From left to right, ϵ threshold value is set to 0.2, 0.5, 0.7 and 1.0. As ϵ increases, edges grow thicker, to the point of darkening the entire rendered image at a maximum of 1.0

seen in equation 3.1 where we use two threshold values for lit outline thickness l and unlit outline thickness u while also making use of light direction $\hat{l}$. Figure 3.3 shows how different parameters for lit and unlit outline thickness can give the rendered images different outline thickness on different areas. Below is an element of the GLSL shader used in which equation 3.1 is used to generate images rendered in figure 3.3. Whenever the conditions for edge detection are fulfilled, edges are drawn in black or else the current vertex corresponds to the interior and thus is drawn in white.

$$0 \leq \hat{n} \cdot \hat{v} \leq mix(u, l, max(0, \hat{n} \cdot \hat{l})) \tag{3.1}$$

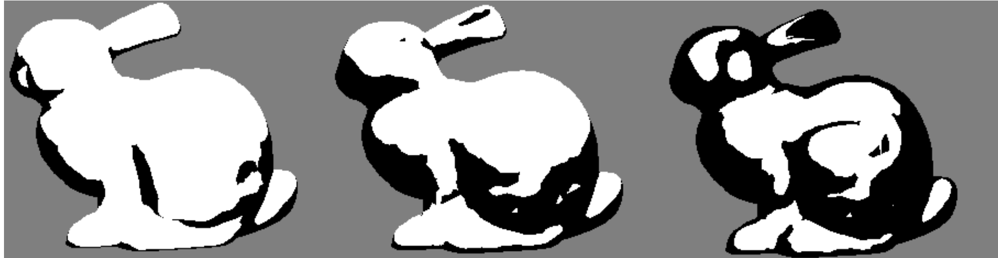

Figure 3.3: Stanford Bunny mesh rendered with different lit and unlit outline edge values. Detected edges are drawn in black whereas interior of the mesh is drawn in white

Drawing of Edges and Strokes

Edges can be easily detected and set to black but in order to get closer matching sumi-e paintings, edges must look as if they were drawn by a paintbrush using sumi. In Chapter 2 a technique which makes use of brush textures is explained with different

brush textures shown in figures 2.6 and 2.7. These brush stroke textures are used within the fragment shader in conjunction with equations 2.2, 2.3 and 2.4 to draw edges with an effect that simulates that of being drawn by a paintbrush. An implementation of this technique is displayed in figure 3.4. Reference code from the GLSL shader is shown in appendix B, listing 2.

Figure 3.4: Drawing of edges with stroke effect using different textures on the Stanford Bunny 3D mesh

Another technique that is of use when working with textures as is the case when drawing edges is that of wobble distortion, used in the work by Charlotte Hoare [15]. Wobble distortion refers to an offset value being given to texture coordinates, as shown in equation 3.2.

$$wobbleUV = uv + uvfsato * wobbleDistortion \tag{3.2}$$

Using wobble distortion allows for texture mapping in edges to be adjusted based on the distortion value as shown in figure 3.5. As can be seen, using wobble distortion changes how edges are drawn without changing textures by adding noise. However, a high value of wobble distortion can lead to edges not matching interior shading properly.

The rendering pipeline also allows for users to draw edges with solid colors, doing so would allow for edges to better match the interior shading when necessary. The implementation currently makes use of only gray scale colors, as shown in figure Code showing both using of textures and solid colors for outlines can be found in 2.

Once edges and strokes are both detected and drawn, the interior of the 3D mesh comes into focus.

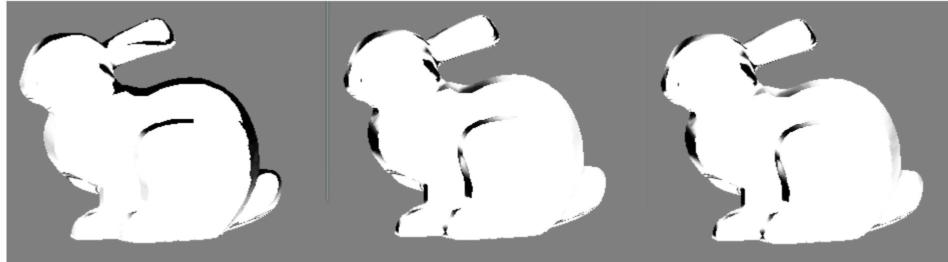

Figure 3.5: Drawing of edges using wobble distortion on the same stroke texture. Edge drawing can be altered slightly based on the stroke texture

Figure 3.6: Drawing of edges using solid colors in gray scale, blending in as the colors get lighter to the point that edges are not seen halfway through and interior edges blend in with the interior as edges are drawn with at their brightest color.

Interior Shading

Techniques used for interior shading such as the use the tone textures used in previous work; as shown in figure 2.9 are one way to draw the interior of a 3D mesh while targeting sumi-e. The rendering pipeline makes use of either a tone texture or a cel shader at the user's discretion.

While using the tone texture, the GLSL shader samples the texture shown in figure 2.9 and equations 2.6, 2.7, 2.8, 2.9 to shade the interior of the 3D mesh. While implementing equation 2.9, the usage of the min function call produced lighter tones to be drawn in areas intended to be of darker shades. Thus, it was changed in favor of clamping values between 0 and 1 as shown in listing 4 which differs from the original implementation shown in listing 3 in appendix B. The original implementation is shown on the left side of figure 3.7 whereas the fixed implementation is shown on the right side.

Similar to the values shown in the tone texture, a cel shader implementation should make use of only black and white colors in order to comply with one of the main

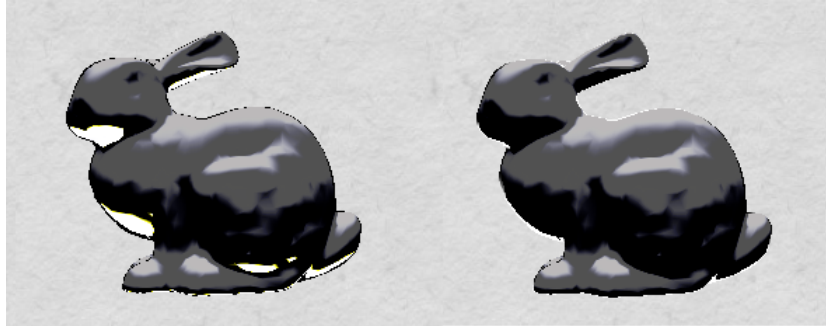

Figure 3.7: Tone texture shading based on the work by Shin-Jin Kang et al [4]. The usage of the tone texture instead of a cel shader allows for a bleeding effect of ink in paper. Left image shows the initial implementation taken from previous work and the right image shows a modified version in which sampled values are clamped. Older implementation code shown in appendix B, list 3 code shown in appendix B, list 4

characteristics of sumi-e paintings. The original tone texture made use of four major colors being black, dark gray, light gray and white and in order to ease the transition between those colors, Gaussian blur was applied. In order to compensate for the lack of a tone texture, intermediate colors for transitioning between black to dark gray, dark gray to light gray and light gray to dark are used as shown in table 3.1

Figure 3.8: Cel-shader used to pick black and white colors based on light intensity on the Stanford Bunny 3D mesh. Code shown in appendix B, list 4

Drawing of 3D model textures

Some 3D meshes might require textures to convey detail either due to the lack of or in addition to detail in the 3D mesh. Both the cel-shader and the tone-texture shader

Color	RGB Value	RGB Value %	Light Intensity
Black	(0, 0, 0)	(0, 0, 0)	< 0.075
Black - Dark Gray 1	(25, 25, 25)	(0.098, 0.098, 0.098)	> 0.075
Black - Dark Gray 2	(59, 59, 59)	(0.2313, 0.2313, 0.2313)	> 0.125
Dark Gray	(82, 82, 82)	(0.321, 0.321, 0.321)	> 0.2
Dark Gray - Light Gray 1	(110, 110, 110)	(0.43, 0.43, 0.43)	> 0.275
Dark Gray - Light Gray 2	(159, 159, 159)	(0.6235, 0.6235, 0.6235)	> 0.325
Light Gray	(188, 188, 188)	(0.737, 0.737, 0.737)	> 0.4
Light Gray - White 1	(204, 204, 204)	(0.8, 0.8, 0.8)	> 0.475
Light Gray - White 2	(227, 227, 227)	(0.89, 0.89, 0.89)	> 0.525
White	(255, 255, 255)	(1.0, 1.0, 1.0)	> 0.6

Table 3.1: Table of color values used by the cel shader implemented in GLSL shader. Results shown in figure 3.8

effect can complement interior textures while rendering images as part of the rendering pipeline implementation.

When using a 3D model texture, its values are sampled and turned to black and white so that colors shown stick to conventions of sumi-e shading by obtaining luminance. Afterwards, color values obtained from either the cel-shader or tone-texture shader are used as diffuse factors which multiply the sampled sampled textur values. A texture luminance factor is used at a user's discretion to increse brightness of the sampled texture. Rendered images of a sphere making use of an orange peel texture are shown in figure 3.9. Example code is shown in 5

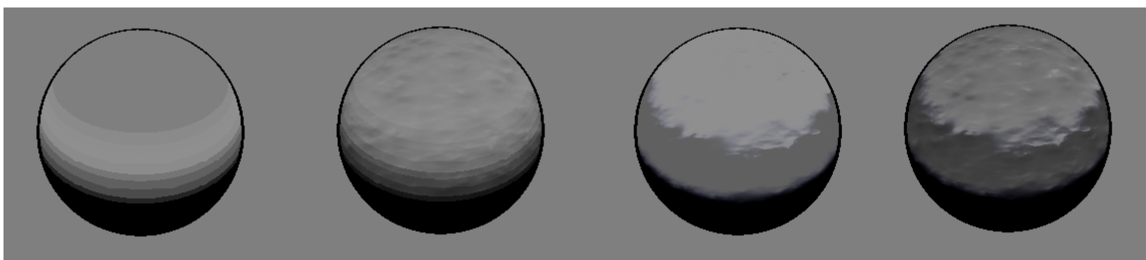

Figure 3.9: Comparison of how a cel-shader and a tone texture shader while making use of interior textures. Code shown in appendix B, list 5

Paper Effect

The techniques researched make use of textures for two main points; the drawing of line edges which is discussed in an earlier section of this chapter and what the work by Kang et al [4] refers to as 'paper effect' by making use of a background texture. As the rendering pipeline developed for this project has a focus on applications for use in video games and is partially inspired in the PlayStation2 video game Ōkami, it was decided to generate a skybox. As mentioned in OGLdev.org [16], a Skybox is a technique which wraps a rendered scene with a texture that goes degrees around the camera. Textures used usually depict scenery such as the sky and mountains. In the case of this project the skybox is a tiling texture of eggshell paper as shown in figure3.10.

Figure 3.10: Texture used for generating the skybox and achieving a paper effect

In order to render the skybox, textures are first loaded with one texture for each face of the skybox being loaded to create a cube map. One the skybox is created it

is rendered before drawing of 3D meshes such as the Stanford bunny 3D mesh. An example of the rendered skybox can be seen in figure 3.11.

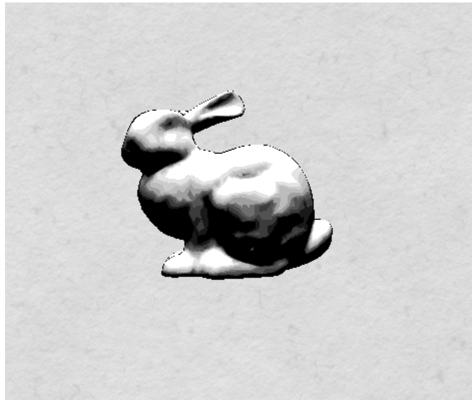

Figure 3.11: Skybox drawn based on the texture shown in fig 3.10 with a drawn mesh of the Stanford bunny

In order to properly replicate the paper texture effect, the background texture needs to be present within the interior of the 3D mesh. To do so the alpha channel is used and modified within the C++ program as well as the GLSL shader. When returning color values for each vertex present in our 3D rendering pipeline, Red, Green and Blue (RGB) color values are specified. In the case of the alpha channel value, the higher; the more opaque/less transparent the vertex is.

In order to make use of transparency effects in the alpha channel, we enable blending and back face culling so that alpha channel values are in use when rendering so that vertices can be transparent while also making it so that back faces are not rendered, avoiding an issue in which back faces are shown as some vertices are transparent, as shown in figure the following shown in listing 6 from appendix B. Enabling alpha channel blending makes it so that output colors are calculated using equation 3.3 which takes current colors and alpha channel values in the frame buffer and obtains new output colors based on the alpha values.

$$OutputDstColor = Alpha * Color + (1 - Alpha) * OutputColor \tag{3.3}$$

As darker colors should blend less with the background, we can infer that the darker the color, the higher the alpha channel is whereas the brighter the color, the more transparent the color is. Thus, we can assign an alpha channel value based on the

average color, meaning that alpha channel is inversely proportional to average color at a vertex, as shown in equation 3.4. An example of how a 3D mesh with and without blending enabled as well as how a mesh looks like with and without back face culling is shown in figure 3.12.

$$AlphaChannel = 1 - (Color.x + Color.y + Color.z)/3 \tag{3.4}$$

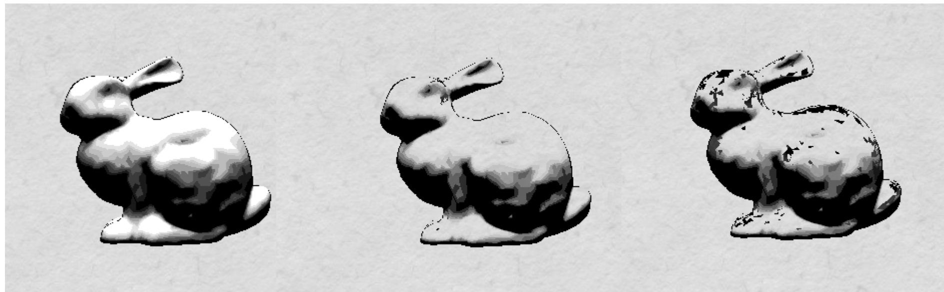

Figure 3.12: Stanford bunny rendered without transparency effects by changing the alpha channel as well as two examples with alpha channel transparency. Middle image makes use of back face culling whereas the right image does not and displays data taken from the back face of the 3D mesh

While equation 3.4 allows us to obtain an alpha value, we make use of another set of parameters in order to force alpha values to be changed and vertices to be drawn or not. First, we make use of an average color threshold value to determine wether to change alpha values or not. If the average color is below a specified threshold, alpha channel value is changed by use of equation 3.4. Another parameter used is an alpha channel threshold value. If the alpha channel is under a specific threshold, vertices are skipped entirely thus allowing for an even more pronounced paper blending effect. Example code is shown in listing 7, appendix B. Examples of how these effects look are shown in figure 3.13. Increasing the average color alpha threshold value makes it so that colors brighter than the threshold value get get their alpha channel value modified according to their average color. The higher the value, the less transparent vertices will be as fewer colors will be of higher intensity than the threshold value. Alpha threshold skip is used to skip/discard vertices whose alpha values are under the threshold value. The higher the alpha threshold skip value, the more vertices are skipped/discarded which makes it so that the 3D mesh blends more with the background with the possibility of

making elements of the back face showing despite backface culling being enabled.

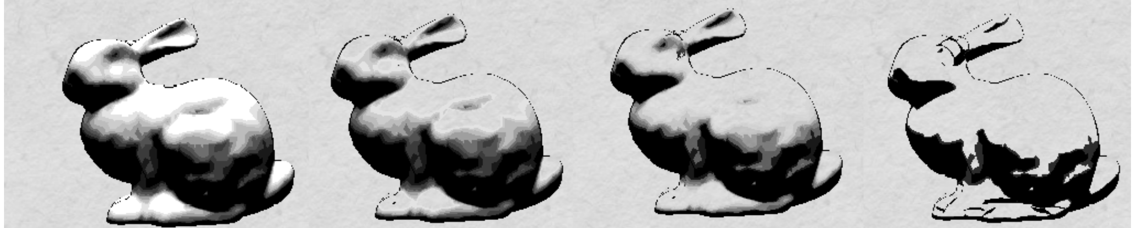

Figure 3.13: Stanford bunny rendered without changes to the average colors alpha threshold as well as the alpha threshold skip values. Code shown in 7, appendix B

3.3 Final Implementation

The final fragment shader for the rendering pipeline is shown in listing 8 and the vertex shader is shown in listing 9, both in appendix B. As various elements of the final program can be fine tuned manually at runtime, they are displayed in a user interface element of the program made with AntTweakbar. UI elements are shown in a small window at the top right corner of the program, shown in figure 3.14 and detailed below:

- Unlit Outline Thickness: changes the thickness of unlit outlines
- Lit Outline Thickness: changes the thickness of lit outlines
- Outline Selection: gives the option of setting a solid edge (false) or using a brush stroke texture of drawing of edges (true)
- Solid Outline Color: when 'Outline Selection' is set to false, allows for the outline's color to be changed in grayscale colors
- Wobble Distortion: when 'Outline Selection' is set to true, allows for offsetting/noise being added to the stroke texture when drawing edges
- Cel Shader Selection: when set to false, a tone texture is used to perform interior shading. When set to true, a cel-shader is used instead

- Texture Selection: when a 3D mesh allows for it and the option is set to true, a texture is used for interior shading in conjunction with the cel shader or tone texture shader
- Texture Luminance: increases the intensity of the tone texture
- Avg Color Alpha Threshold: changing this value allows for alpha channel values to be changed based on a color's average value. The higher the average color threshold value, the more colors get alpha channel changed.
- Alpha Threshold Skip: used to skip vertices whose alpha channel values are lower than or equal to the threshold value. The highest value available can make the 3D model disappear as all vertices are discarded

In addition to the parameters shown in AntTweakbar for changing parameters in the rendering pipeline, keyboard inputs can be used to move around the problem, shown below.

- w - moves camera position up
- a - moves camera position down
- s - moves camera position left
- d - moves camera position right
- mouse wheel forward - moves camera position forward
- mouse wheel backwards - moves camera position backwwards
- l - moves light source around the environment
- r - rotates 3D mesh on its axis
- R - resets keyboard input parameters (camera position, 3D mesh rotation, light source position)

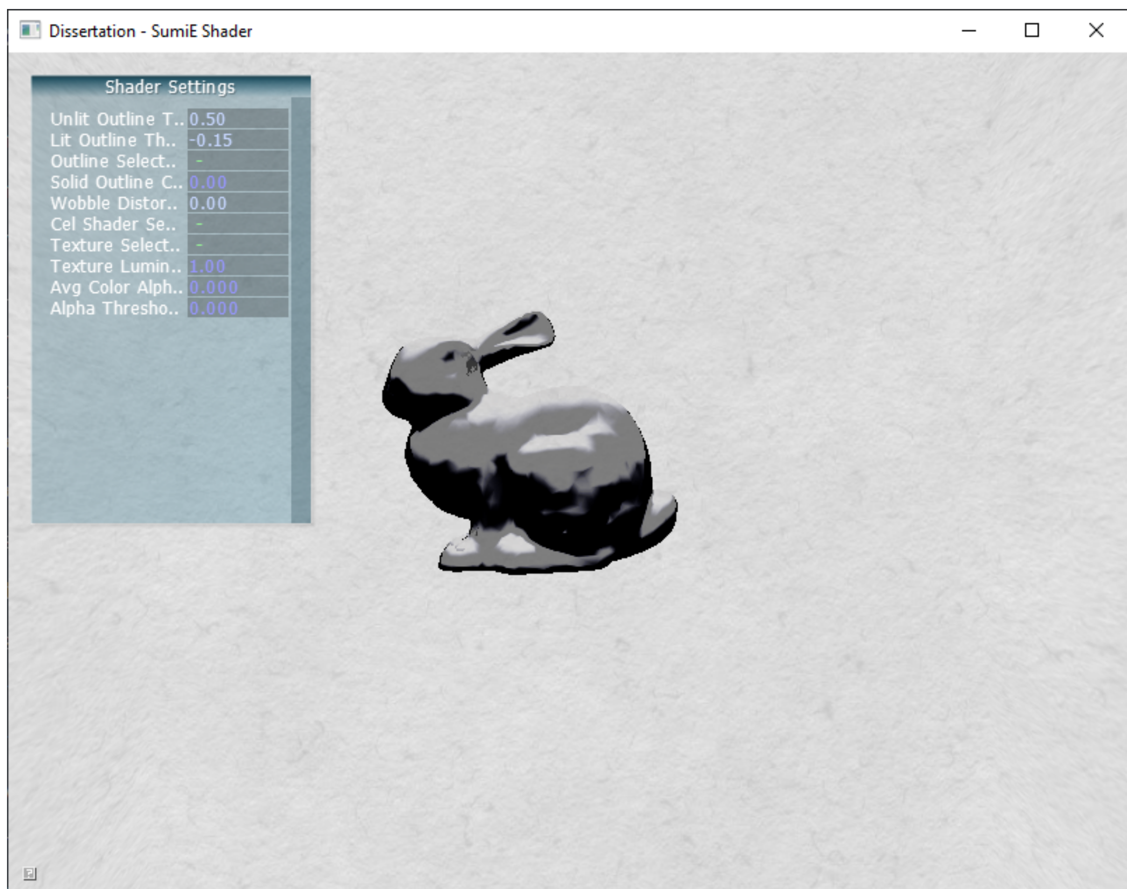

Figure 3.14: User interface elements added with AntTweakbar for manipulating behavior of the rendering pipeline

Chapter 4

Results

This chapter covers rendered images obtained by using the final rendering pipeline and evaluate how close they match characteristics of sumi-e paintings. A variety of 3D models will be used to verify the flexibility of the rendering pipeline while working with different models and parameters. Performance will also be measured in terms of frame rate to evaluate the likelihood of a sumi-e style rendering pipeline being used for real-time rendered media such as video games. A conclusion will be given on whether a rendering pipeline akin to the one developed for this project could be usable in media such as video games.

4.1 Tests and Comparison with Sumi-E

As shown in figure 3.1, the rendering pipeline consists of multiple effects which can be used to modify the look of a 3D model so that it might match characteristics of sumi-e paintings. As previously established, characteristics of sumi-e paintings the rendering pipeline and resulting images should target are:

- Usage of only black and white colors
- Usage of ink (sumi) as a medium and egg shell paper (torinoko) as a canvas with a brush as an applicator
- Usage of long/broad brush strokes

- Lack of sketching/drawing of outlines done before usage of the paint brush

Figure 4.1 displays different angles of the Stanford bunny drawn using the final rendering pipeline with varying degrees of outline thickness and using either the cel shader or the tone texture shader as options. Characteristics such as the use of black and white colors, usage of brush strokes and lack of previous sketching can be seen as well as the use of eggshell paper due to the eggshell paper skybox in addition to transparency via use of the alpha channel.

Figure 4.1: Rendering pipeline output using the Stanford bunny on different positions with variation in outline thickness and usage of cel-shader or tone texture shader

While specific characteristics of sumi-e painting are achieved by the rendering pipeline, there are elements that struggle when trying to match the targeted art style. Using brush stroke textures with sphere mapping can achieve a brush stroke look while drawing edges as shown on the rightmost rendered image in figure 4.1, however it can also be inconsistent with how the interior of a 3D mesh is drawn and using a high wobble distortion value can lead to edges being drawn in way that appears as if edges were drawn/sketched previous to painting, conflicting with one of the major characteristics of sumi-e, as shown in figure 4.2.

Figure 4.2: Rendered images of the Stanford bunny using badly picked values for edges drawn with stroke textures using sphere mapping

In order to achieve the paper effect, the rendering pipeline makes use of the alpha channel to make elements of the image transparent and blend with the background texture that makes up the skybox/canvas. To do so, a sampled vertex makes use of its average color to obtain the alpha channel value which is inversely proportional to the average color. Drawing objects with alpha transparency effects requires back face culling to be enabled so that faces are not drawn as artifacts from the back face can show up as seen in figure 3.12.

Enabling back face culling allows for the background texture taken from the skybox to complement interior shading of a 3D mesh. However, upon doing a close inspection as seen in figure 4.3, back face culling can have inconsistent results as elements of the feet and ears fromt he back face of the Stanford bunny can be seen at the front face despite back face culling being enabled. Adjustments to parameters of the rendering pipeline such as using a solid yet lighter color for drawing edges, outline thickness changes or different textures used for drawing edges can help remove elements from the back face being shown through the front face of a 3D mesh, as shown in figure 4.3. Issues with back face culling when a mesh interior is fully transparent can be more troublesome in a game environment when elements of the scene are directly behind others. Figure 4.4 shows how the issue can look when back face culling is not enabled (left) and when back face culling is enabled (right) while much of the interior shading is transparent.

Another 3D model used for testing the rendering pipeline is that of a bamboo tree taken from TurboSquid [17]. The bamboo tree consists of a stalk and a high density of leaves at the top, includes textures for the 3D model. Figure 4.5 displays the mesh rendered with and without alpha transparency effects while using the cel shader and no texture obtained from the original source used for interior shading. As can be seen, transparency effects allow for the mesh to blend in with the background when shown from afar and even when not having transparency effects enabled, a mesh such as the bamboo tree can blend in as part of the scenery.

Figure 4.6 displays multiple closeups of the bamboo tree, alternating between using the cel-shader or the tone texture shader whilst also enabling and disabling the bamboo clum texture included with the 3D mesh used for interior shading. These closeups allow for a better comparison with the example painting shown in figure 2.1 which displays a bamboo stalk in conjunction with leaves. Gaps between sections of the stalk can be

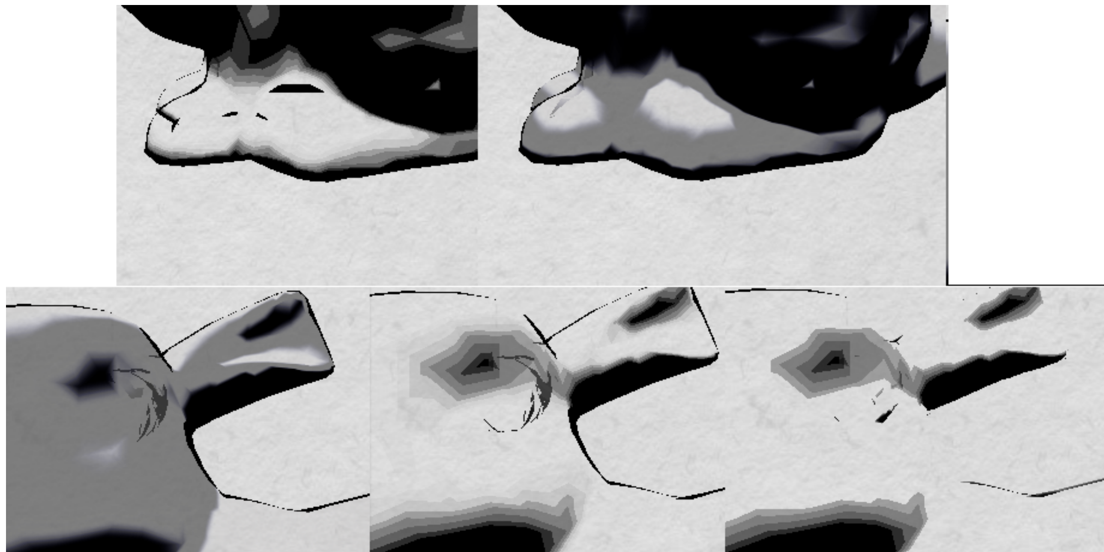

Figure 4.3: Closeup of the Stanford Bunny 3D mesh when using alpha transparency effects. Shows back face culling inconsistencies and how slight variations in parameters can help diminish the issue when present

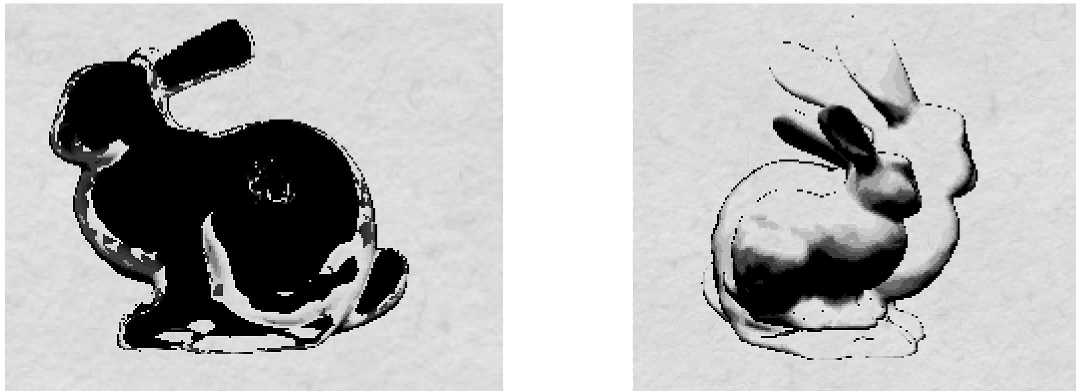

Figure 4.4: Stanford Bunny 3D mesh drawn in both the background and foregound, without and with face culling enabled

seen in all rendered images and shading of leaves can match the original painting in certain areas. A closer look at leaves rendered and compared with the original painting can be seen in figure 4.7, where using the tone-texture shader makes rendered images of leaves match the painting shown in figure 2.2 more closely than the cel-shader.

The rendering pipeline may be used with objects that deviate from typical subjects of sumi-e; although there might be varying degrees of success with certain 3D models

Figure 4.5: Rendered image of a bamboo tree drawn using the cel-shader from the rendering pipeline. Left side shows no alpha transparency effects added. Right side shows alpha transparency effects

while attempting to match sumi-e paintings. As an example, a sphere mesh generated using Blender in conjunction with an orange peel texture obtained from [7], shown in figure 4.8, the usage of textures for shading of a model's interior can be observed. When using the tone-texture shader, transparency effects are not an issue compared to the Stanford Bunny 3D mesh. Using a texture for interior shading allows detail to be shown in the mesh despite the texture being turned to black and white, still the low vertex count of the sphere mesh leads to issues with interior shading, particularly when making use of cel-shading for drawing the interior.

Figure 4.10 makes use of a smoother 3D sphere mesh which has a higher vertex count. This 3D sphere displays the effects of the sumi-e shader better. The cel-shader achieves a smoother look but it is still not ideal as the interior shading makes the fact that the sphere is made up of vertices apparent, even if it is not as apparent as in the previous version of the sphere. Despite the smoother 3D mesh, the cel-shader does not produce as natural interior shading as the tone-texture shader, whether it is using interior textures or not or using transparency effects or not. Other effects

Figure 4.6: Closeup rendered images of the bamboo tree. Top section makes use of the tone-texture shader and bottom texture makes use of the cel-shader. Left section makes no use of textures for interior shading and right section makes use of textures for interior shading

Figure 4.7: Closeup to images of bamboo leaves. Left image is taken from 2.2 whereas center and right image are taken from 4.6. Center image makes use of tone texture shading whereas the right image makes use of the cel-shader

such as transparency using alpha channel values and the drawing of edges look cleaner when using both the cel-shader and the tone texture shader effect. Tone texture in

conjunction with in the orange peel used for shading the interior achieves a more detailed look and a bleeding effect not seen when using the cel-shader on the same model.

Figure 4.8: Orange texture taken from [7].

One example mesh which deviates from sumi-e is that of a cartoon styled, low-poly airplane obtained from TurboSquid [8], shown in figures 4.11 and 4.12. The initial model has a total of 1018 vertices, half of which correspond to the landing wheels and propellers. As can be seen in figure 4.11, both the cel-shader and tone texture shader have issues shading the interior evenly. Transparency effects used to convey the use of paper also show back faces despite back face culling being enabled when using the cel-shader. Figure 4.12 shows the same 3D mesh modified by adding additional vertices in blender, having a much higher vertex count of 855275. Despite this, effects do not apply evenly to the updated version and none of the rendered images achieve a look that matches sumi-e beyond the use of black and white colors.

4.2 Performance

As the hardware used for this project is newer and more powerful than that used by previous works, particularly compared to the PlayStation2 which was Ōkami's target platform; performance was high and consistent across all tests done, reaching frame rates over 144 frames per second even on cases in which the vertex count was much higher, as can be seen in table 4.1.

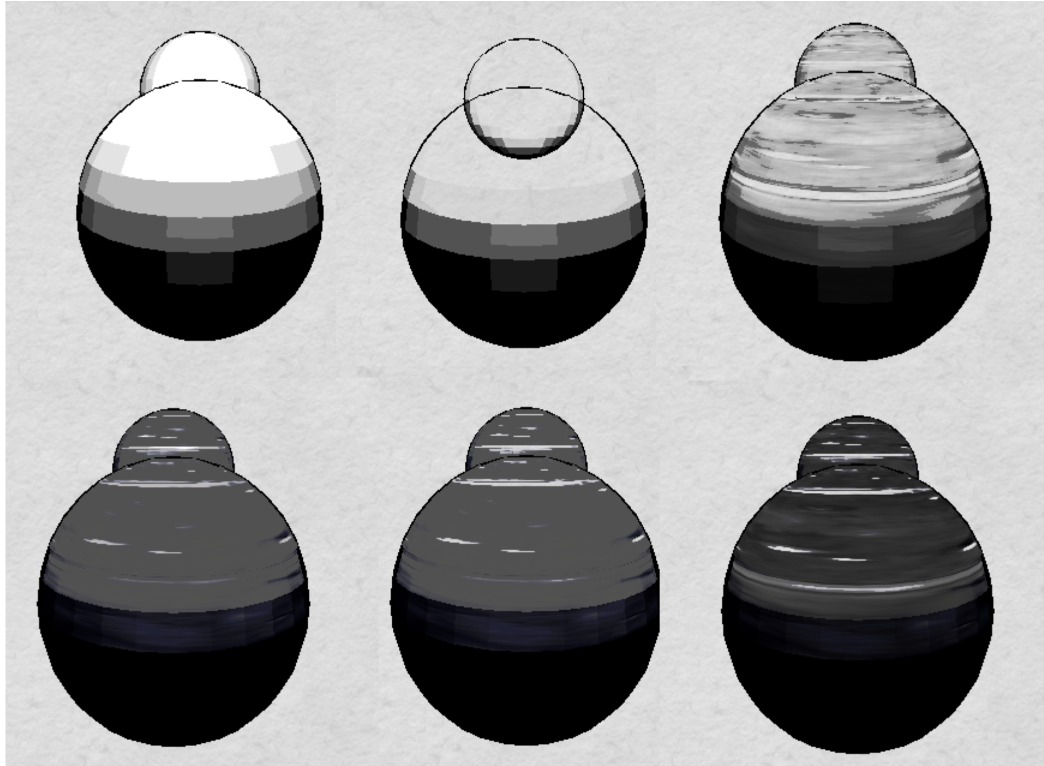

Figure 4.9: Sphere mesh generated using Blender. Top section makes use of the cel-shader whereas bottom section makes use of the tone-texture shader. Left section makes no use of transparency/alpha effects, middle section makes use of transparency/alpha effects and right section makes use of transparency/alpha effects in conjunction with an orange peel texture taken from [7]

4.3 Conclusions

There is no one-size fits all approach when using the rendering pipeline developed for this project. The ability of the rendering pipeline to match the art style of sumi-e can be affected by modifying parameters that require user input such as outline thickness, use of stroke textures or the type of interior shader. Data from the rendered model such as the shape, complexity of the model as well as the corresponding textures also affect the resulting images. For example, models such as the Stanford Bunny or a bamboo tree can match the art style of sumi-e when using the rendering pipeline properly.

There are occasions in which a 3D model might not be suited for using sumi-e as its target art style, as was the case of the cartoon style airplane shown in figures 4.12 and

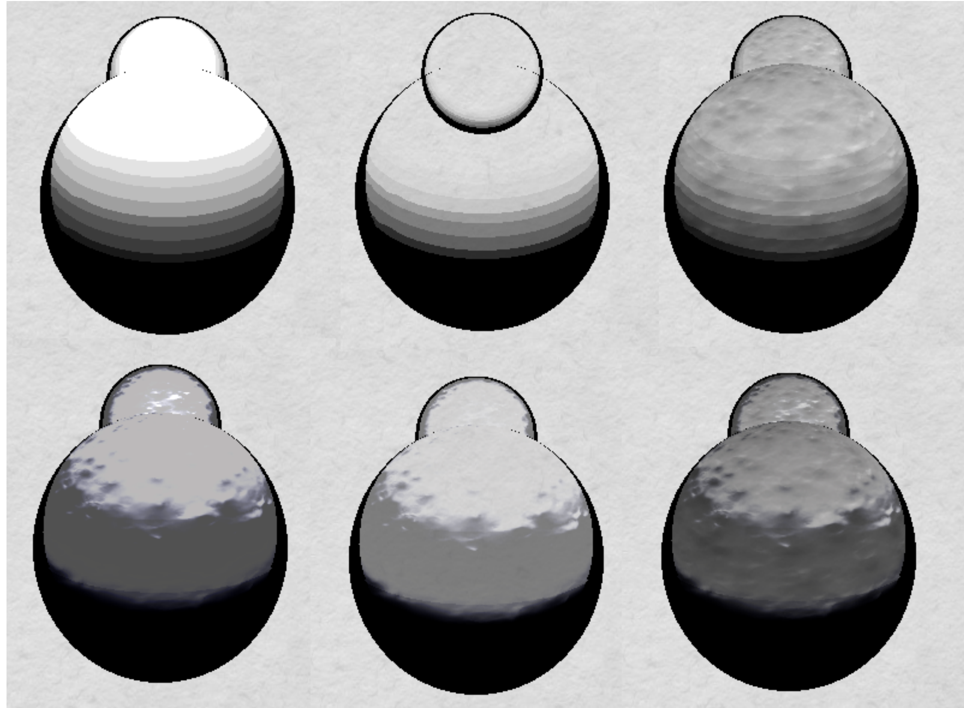

Figure 4.10: Similar sphere to that of figure 4.9, using a higher vertex count mesh made using Blender

4.11 taken from [8]. A low polygon model or a model with a multitude of flat surfaces can have difficulties matching sumi-e although using interior textures as shown in figure 4.9 can make a low-poly model more closely match the targeted art style.

Certain effects are well suited for generating sumi-e style images but can lead to issues in a video-game environment as is the case of the paper effect when using alpha transparency without a tone texture shader or interior texture, as can be seen in figure 4.3 as well as figure 4.4 since background models and elements of a model's back face might lead to issues making out characters. Comparing the results to Ōkami, despite targeting sumi-e as its desired art style, it deviated in a multitude of areas.

Color was prevalent in the game and not all on-screen elements made use of the same paper effect, making the game's output less consistent with sumi-e while also being better suited for a real-time game. It also made use of color which allowed for users to better differentiate between rendered objects, as shown in figure 1.1.

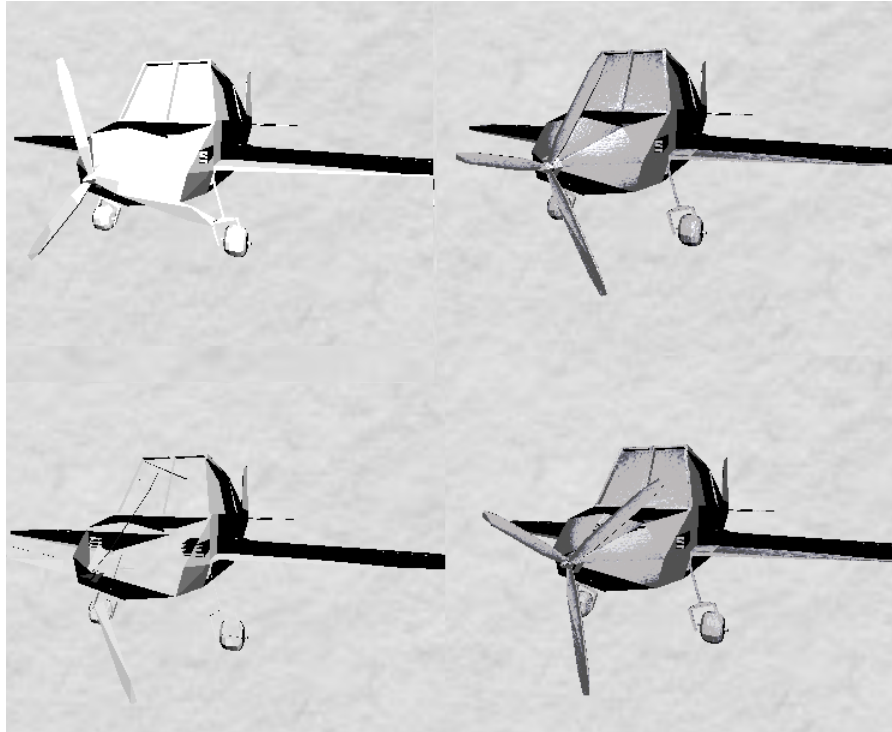

Figure 4.11: Low-poly airplane 3D model from [8] rendered with a cel-shader (left) and a tone-texture shader (right) with transparency effects disabled (top) and enabled(bottom)

Mesh	Vertices	Framerate
Stanford Bunny	2503	144
Sphere	482	144
Smooth Sphere	58082	144
Cartoon Airplane	1018	144
Smooth Cartoon Airplane	855275	144
Bamboo Tree	73859	144

Table 4.1: Table displaying performance for each experiment run when testing the sumi-e rendering pipeline. Tests were done rendering images at a resolution of 1024 x 768

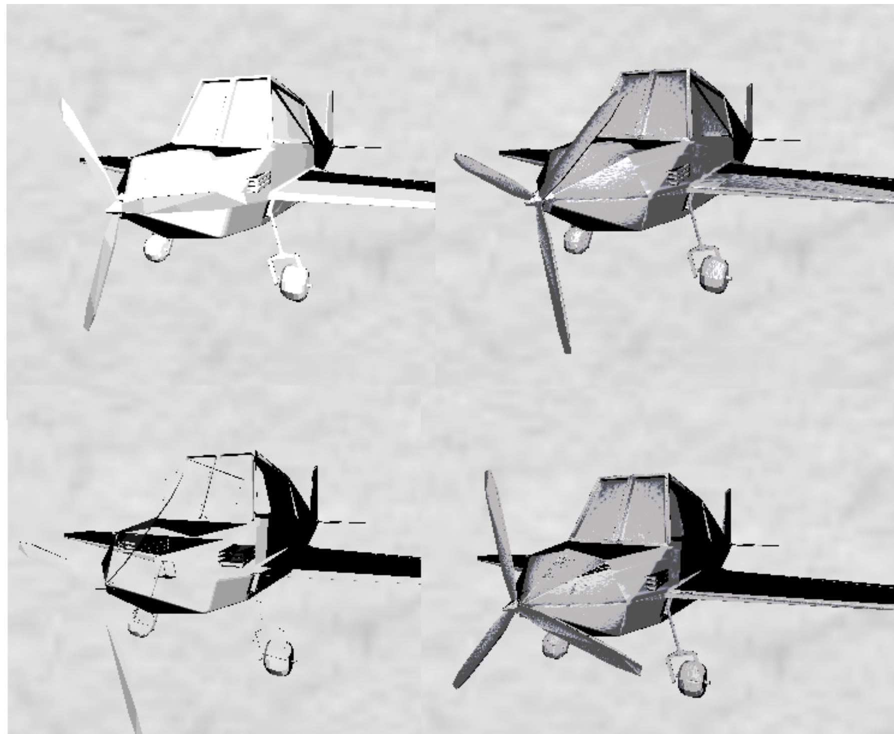

Figure 4.12: Modified low-poly airplane 3D model from [8], edited in Blender to have a much higher vertex count. Rendered with a cel-shader (left) and a tone-texture shader (right) with transparency effects disabled (top) and enabled(bottom)

Chapter 5

Future Work

The rendering pipeline developed for this project can be used for generating sumi-e style images, however; an application such as video-games would require further research. As performance in all experiments was high despite using a higher resolution compared to experiments from previous works, it can be assumed that the rendering pipeline would translate well into a game environment, performance wise. However, this can be due to the fact that the hardware used for this project was newer and relatively more powerful than hardware used in previous efforts.

Because of this, further research should be done by implementing a sumi-e rendering pipeline in a more complex environment such as a game engine, testing how the rendering pipeline would affect performance while working with more complex 3D models, higher resolution textures and bigger environments. Complex animations and interactions between 3D models such as collisions should also be tested to verify the likelihood of sumi-e being an adequate target art style for video games.

Using a more feature complete tool set such as a game engine can also be of use to creating animations, deviating from video games as interactions between 3D models can apply to both real-time and offline rendering applications. Offline animation also has the possibility of being better suited for this targeted art style due to the nature of the medium. As such, further research should be done testing how sumi-e works as a targeted art style for offline animation.

Bibliography

[1] H. P. Bowie, *On the Laws of Japanese Painting: An Introduction to the Study of the Art of Japan.* San Francisco P. Elder and Company, 1911. original-date: 1911.

[2] S. Strassmann, "Hairy brushes," *ACM SIGGRAPH Computer Graphics*, vol. 20, pp. 225–232, Aug. 1986.

[3] B. Pham, "Expressive brush strokes," *CVGIP: Graphical Models and Image Processing*, vol. 53, pp. 1–6, Jan. 1991.

[4] S.-J. Kang, S.-J. Kim, and C.-H. Kim, "Hardware-accelerated real-time rendering for 3d sumi-e painting," in *Computational Science and Its Applications — ICCSA 2003* (G. Goos, J. Hartmanis, J. van Leeuwen, V. Kumar, M. L. Gavrilova, C. J. K. Tan, and P. L'Ecuyer, eds.), vol. 2669, pp. 599–608, Berlin, Heidelberg: Springer Berlin Heidelberg, 2003.

[5] J.-H. Park, S.-J. Kim, C.-G. Song, and S.-J. Kang, "Hardware-Accelerated Sumi-e Painting for 3D Objects," in *Computational Science – ICCS 2009* (G. Allen, J. Nabrzyski, E. Seidel, G. D. van Albada, J. Dongarra, and P. M. A. Sloot, eds.), vol. 5545, pp. 780–789, Berlin, Heidelberg: Springer Berlin Heidelberg, 2009.

[6] A. Hutchins and S. Kim, "Advanced Real-Time Cel Shading Techniques in OpenGL." "https://www.cs.rpi.edu/~cutler/classes/advancedgraphics/S12/final_projects/hutchins_kim.pdf", 2012. [online] Accessed: 21-Jun-2020.

[7] Moss, "Citrus." "https://www.filterforge.com/filters/7375.html", 2009. [online] Accessed: 02-Sep-2020.

[8] A. Moek, "Cartoon low poly airplane 3d." "https://www.turbosquid.com/3d-models/plane-render-3d-1192348", 2017. [online] Accessed: 28-Aug-2020.

[9] R. Valentine, “Sony invests $250m in Epic Games.” “https://www.gamesindustry.biz/articles/2020-07-09-sony-invests-USD250m-in-epic-games”, 2020. [online] Accessed: 12-Jul-2020.

[10] B. Gooch and A. Gooch, *Non-photorealistic rendering.* Natick, Mass: A K Peters, 2001.

[11] GamesIndustryInternational, “Oart and mythology come alive as Capcom® announces Okami for the playstation 2 computer entertainment system.” “https://www.gamesindustry.biz/articles/oart-and-mythology-cme-alive-as-capcom-announces-okami-for-the-playstation-2-computer-entertainment-system”, 2005. [online] Accessed: 21-Jun-2020.

[12] T.-C. Xu, L.-J. Yang, and E.-H. Wu, “Stroke-based real-time ink wash painting style rendering for geometric models,” in *SIGGRAPH Asia 2012 Technical Briefs on - SA ’12*, (Singapore, Singapore), pp. 1–4, ACM Press, 2012.

[13] “The Stanford 3d Scanning Repository.” “https://graphics.stanford.edu/data/3Dscanrep/”, 2014. [online] Accessed: 18-Jun-2020.

[14] WikiBooks.org, “Glsl programming/unity/toon shading - wikibooks, open books for an open world.” “https://en.wikibooks.org/wiki/GLSL_Programming/Unity/Toon_Shading#Stylized_Diffuse_Illumination”, 2011. [online] Accessed: 28-Aug-2020.

[15] C. Hoare, “Interactive non-photorealistic rendering using glsl.” “https://nccastaff.bournemouth.ac.uk/jmacey/MastersProjects/MSc12/Hoare/CharlotteHoare_Thesis.pdf”, 2012. [online] Accessed: 19-Aug-2020.

[16] OGLDev.org, “Tutorial 25 - skybox.” “http://www.ogldev.org/www/tutorial25/tutorial25.html”, 2011. [online] Accessed: 13-Jul-2020.

[17] IndaneeyDesign, “3d bamboo tree.” “https://www.turbosquid.com/3d-models/3d-tree-nature-forest-1421571”, 2019. [online] Accessed: 16-Aug-2020.

Appendix A - Acronyms

Acronym	**Definition**
3D	Three-Dimensional
NPR	Non-Photorealisctic Rendering
GPU	Graphics Processing Unit
GLSL	OpenGL Shading Language
OpenGL	Open Graphics Library
RGB	Red, Green, Blue

Appendix B - Code

Listing 1: Edge detection with outline thickness

```
1  // Lit/unlit edge detection and drawing
2  if (dot(viewDir, normal_dir) < mix(unlit_outline_thickness,
      lit_outline_thickness, max(0.0, dot(normal_dir, light_dir))))
3  {
4   color = vec4(vec3(0.0, 0.0, 0.0), 1.0);
5  } else {
6  // Not an edge, drawn in white
7   color = vec4(vec3(1.0, 1.0, 1.0), 1.0);
8  }
9  // Returnin color
10 frag_color = color;
```

Listing 2: Edge detection with outline thickness in conjunction with texture mapping and texture wobbling

```
1      // Determining wether a vertex is an edge or not and its outline
          thickness
2   if (dot(viewDir, normal_dir) < mix(unlit_outline_thickness,
       lit_outline_thickness, max(0.0, dot(normal_dir, light_dir)))) {
3    if(outline_selection == true){
4    // Drawing edge with brush texture and wobble distortion
5      color = clamp(texture(brush_texture_map, vec2(tu, tv) + vec2(tu, tv)*
          wobble_distortion), 0, 1);
6    }else{
7    // Drawing edge with a solid color
8      color = vec4(solid_outline_color, solid_outline_color,
          solid_outline_color, 1.0);
9    }
10  }
```

Listing 3: GLSL shader segment with option for using cel shader or a tone texture shader. Makes use of min functoin when doing tone texture shading

```
1  if(cel_shader_selection == true){
2              // Cel shader code
3  }else{
4   vec3 diffuse_texture_color = texture(mesh_texture_map,
        a_texture_coordinate).rgb;
5   float tone_texture_u = min(intensity*(0.3*diffuse_texture_color.r +
6                0.59*diffuse_texture_color.g +
7                0.11*diffuse_texture_color.b),
8              1);
9   color = texture(tone_texture_map, vec2(tone_texture_u, 0.5));
10 }
```

Listing 4: GLSL shader segment with option for using a cel shader or a tone texture shader. Uses clamping of clamping of sampled color for obtaining texture coordinates

```
1  if(cel_shader_selection == true){
2   // Cel shader code
3  }else{
4   vec3 diffuse_texture_color = texture(mesh_texture_map,
        a_texture_coordinate).rgb;
5   float tone_texture_u = clamp(intensity*(0.3*diffuse_texture_color.r +
6               0.59*diffuse_texture_color.g +
7               0.11*diffuse_texture_color.b),
8              0, 1);
9   color = texture(tone_texture_map, vec2(tone_texture_u, 0.5));
10 }
```

Listing 5: GLSL shader segment showing how 3D model textures can be used in addition to a cel shader or a tone texture shader

```
1 if(texture_selection==true){
2  // Normal Mapping Part
3  vec3 diffuse_texture_color = texture(mesh_texture_map,
       a_texture_coordinate).rgb;
4  // Limunance
5  float diffuse_texture_bw = texture_luminance*dot(diffuse_texture_color,
       vec3(0.2126729, 0.7151522, 0.0721750));
6
```

```
7    // cel shader color * diffuse texture BW color
8     color = color*vec4(diffuse_texture_bw, diffuse_texture_bw,
        diffuse_texture_bw, 1.0);
9   }
```

Listing 6: main.cpp segment showing the enabling of alpha channel blending and back face culling

```
1   glEnable(GL_BLEND);
2   glBlendFunc(GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA);
3   glEnable(GL_CULL_FACE);
4   glCullFace(GL_BACK);
```

Listing 7: GLSL shader segment showing alpha channel enabling based on average color threshold and discarding of vertices via an alpha channel threshold value

```
1   // Average color used to obtain alpha value
2   float avg_color = clamp((color.x + color.y + color.z)/3, 0.0, 1.0);
3   if(avg_color > avg_color_alpha_threshold){
4    color.a = clamp((1 - avg_color), 0.0, 1.0);
5   }
6
7   // Threshold valuesfor discarding pixels
8   if(color.a < alpha_skip_threshold){
9    discard;
10  }
```

Listing 8: Final fragment shader used for the rendering pipeline

```
1  #version 330
2
3  in vec3 n_eye;
4  in vec3 pos_eye;
5  in vec3 vs_normals;
6  in vec3 vs_position; // vs_position
7  in vec3 vs_position_iso; // zwvEcVertex
8  in vec2 a_texture_coordinate;
9
10 uniform sampler2D brush_texture_map;
11 uniform sampler2D mesh_texture_map;
12 uniform sampler2D tone_texture_map;
```

```
13
14 uniform samplerCube cube_texture;
15
16
17 out vec4 frag_color;
18
19 uniform mat4 model, ortho, proj, view;
20 uniform vec3 lightPos, viewPos;
21 uniform float ambientStrength, specularStrength;
22  // Turn to uniforms later
23 uniform float lit_outline_thickness, unlit_outline_thickness,
       solid_outline_color, wobble_distortion, texture_luminance,
       paper_alpha_threshold, paper_alpha_div;
24 uniform bool outline_selection, texture_selection, cel_shader_selection;
25 // Rename/delete these
26 uniform float diffuse_factor, dry_brush_granulation, dry_brush_density;
       // Make this uniform variable (?)
27
28 const vec3 ambientColor = vec3(0.0, 1.0, 0.0);
29 const vec3 diffuseColor = vec3(1.0, 1.0, 1.0);
30 const vec3 specColor = vec3(0.5, 0.5, 0.5);
31 const vec2 resolution  = vec2(1024, 768);
32
33 void main() {
34  // Global Lighting Variables
35  vec4 color;
36  vec3 light_dir = normalize(lightPos - pos_eye);
37  vec3 norm = normalize(vs_normals);
38  vec3 viewDir = normalize(viewPos - pos_eye);
39  // Toon
40  vec3 normal_dir = normalize(vs_normals);
41  vec3 outLineColor = vec3(0.0, 0.0, 0.0);
42  vec3 lightColor = vec3(1.0, 1.0, 1.0);
43  float intensity;
44  intensity = dot(light_dir, normalize(norm));
45  // Sumi-E Edges
46  vec3 vee = viewPos - vs_position;
47  vec3 vee_norm = normalize(vee);
48  vec3 norm_view = (view * vec4(vs_normals, 1.0)).xyz;
```

```
49  vec3 r = 2*(dot(norm_view, vee_norm))*(norm_view-vee_norm);
50  float m = 2 * sqrt(pow(r.x, 2)+pow(r.y, 2) + pow(r.z+1,2));
51  float tu = r.x/m + 1/2;
52  float tv = r.y/m + 1/2;
53
54
55  // if (dot(viewDir, normal_dir) < 1.0) {
56  if (dot(viewDir, normal_dir) < mix(unlit_outline_thickness,
       lit_outline_thickness, max(0.0, dot(normal_dir, light_dir)))) {
57   if(outline_selection == true){
58    color = clamp(texture(brush_texture_map, vec2(tu, tv) + vec2(tu, tv)*
         wobble_distortion), 0, 1);
59   }else{
60    color = vec4(solid_outline_color, solid_outline_color,
         solid_outline_color, 1.0);
61   }
62
63  } else {
64   if(cel_shader_selection == true){
65    // Main color W
66    if (intensity > 0.6) {
67     color = vec4(1.0, 1.0, 1.0, 1.0);
68    }
69    // Transition from secondary colors 3
70    // Secondary color LG-W2
71    else if (intensity > 0.525) {
72     color = vec4(0.89, 0.89, 0.89, 1.0);
73    }
74    // Secondary color LG-W1
75    else if (intensity > 0.475) {
76     color = vec4(0.8, 0.8, 0.8, 1.0);
77    }
78    // Main color LG
79    else if (intensity > 0.4) {
80     color = vec4(0.737, 0.737, 0.737, 1.0);
81    }
82    // Transition from secondary colors 2
83    // Secondary color DG-LG2
84    else if (intensity > 0.325) {
```

```
85      color = vec4(0.6235, 0.6235, 0.6235, 1.0);
86     }
87     // Secondary color DG-LG1
88     else if (intensity > 0.275) {
89      color = vec4(0.43, 0.43, 0.43, 1.0);
90     }
91     // Main color DG
92     else if (intensity > 0.2) {
93      color = vec4(0.321, 0.321, 0.321, 1.0);
94     }
95     // Transition from secondary colors 1
96     // Secondary color B-DG2
97     else if (intensity > 0.125) {
98      color = vec4(0.2313, 0.2313, 0.2313, 1.0);
99     }
100    // Secondary color B-DG1
101    else if (intensity > 0.075) {
102     color = vec4(0.098, 0.098, 0.098, 1.0);
103    }
104    // Main color B
105    else {
106     color = vec4(0.0, 0.0, 0.0, 1.0);
107    }
108   }else{
109    vec3 diffuse_texture_color = texture(mesh_texture_map,
          a_texture_coordinate).rgb;
110    // float tone_texture_u = min(intensity*(0.3*diffuse_texture_color.r +
111    //              0.59*diffuse_texture_color.g +
112    //              0.11*diffuse_texture_color.b),
113    //            1);
114    float tone_texture_u = clamp(intensity*(0.3*diffuse_texture_color.r +
115                 0.59*diffuse_texture_color.g +
116                 0.11*diffuse_texture_color.b),
117               0, 1);
118    color = texture(tone_texture_map, vec2(tone_texture_u, 0.5));
119   }
120  }
121  if(texture_selection==true){
122   // Normal Mapping Part
```

```
123   vec3 diffuse_texture_color = texture(mesh_texture_map,
          a_texture_coordinate).rgb;
124   // Limunance
125   float diffuse_texture_bw = texture_luminance*dot(diffuse_texture_color,
          vec3(0.2126729, 0.7151522, 0.0721750));
126
127   // cel shader color * diffuse texture BW color
128   color = color*vec4(diffuse_texture_bw, diffuse_texture_bw,
          diffuse_texture_bw, 1.0);
129  }
130
131  // Average color used to obtain alpha value
132  float avg_color = clamp((color.x + color.y + color.z)/3, 0.0, 1.0);
133  if(avg_color <= paper_alpha_threshold){
134   color.a = clamp((1 - avg_color), 0.0, 1.0);
135  }
136
137  // Threshold valuesfor discarding pixels
138  if(color.a < paper_alpha_div){
139   discard;
140  }
141  frag_color = color;
142 }
```

Listing 9: Final vertex shader used by the rendering pipeline

```
1  #version 330
2
3  //layout(location = 0) in vec3 vertexPosition;
4  //layout(location = 1) in vec3 vertexNormals;
5  in vec3 vertexPosition;
6  in vec3 vertexNormals;
7
8  out vec3 nEye;
9  out vec3 fragPos;
10 out vec3 vsNormals;
11 out vec3 vsPosition;
12
13 uniform mat4 model, ortho, proj, view;
14 uniform vec3 lightPos, viewPos;
```

```
15 uniform float ambientStrength, specularStrength;
16
17 void main(){
18  fragPos = vec3(model * vec4(vertexPosition, 1.0));
19  nEye = (view * vec4 (vertexNormals, 0.0)).xyz;
20  //vsNormals = vertexNormals;
21  vsNormals = normalize(mat3(model) * vertexNormals);
22  //vsPosition = vertexPosition;
23  vsPosition = mat3(model) * vertexPosition;
24  gl_Position =  proj * view * model * ortho* vec4 (vertexPosition, 1.0);
25 }
```