Real-Time Hatch Rendering

Luyao Wang

A thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Computer Science & Software Engineering

University of Washington

2022

Reading Committee:

Kelvin Sung, Chair

Yusuf Pisan

Michael Stiber

Program Authorized to Offer Degree:

Computer Science and Software Engineering

University of Washington

Abstract

Real-Time Hatch Rendering

Luyao Wang

Chair of the Supervisory Committee:
Dr. Kelvin Sung
Computer Science and Software Engineering

Hatching has been a common and popular artistic drawing style for centuries. In computer graphics rendering, hatching has been investigated as one of the many Non-Photorealistic Rendering solutions. However, existing hatch rendering solutions are typically based on simplistic illumination models, and real-time 3D hatch-rendered applications are rarely seen in interactive systems such as games and animations. This project studies the existing hatch rendering solutions, identifies the most appropriate one, develops a real-time hatch rendering system, and improves upon existing results in three areas: support general illumination and hatch tone computation related to observed artistic styles, unify spatial coherence support for Tonal Art Maps and mipmaps, and demonstrate support for animation.

The existing hatch rendering solutions can be categorized into texture-based and primitive-based methods. These solutions can be derived in object or screen space. Based on our background research, we chose to examine the texture-based object-space method presented by Praun et al. The approach inherits the advantage of object-space temporal coherence. The object-space spatial incoherence is addressed by the introduction of the Tonal Art Map (TAM). The texture-based solution ensures that the rendering results resemble actual artists' drawings.

The project investigated the solution proposed by Praun et al. based on two major components: TAM generation as an off-line pre-computation and real-time rendering via a Multi-Texture Blending shader.

The TAM construction involves building a two-dimensional structure, vertically to address spatial coherence as projected object size changes and horizontally to capture hatch tone changes. This unique structure enables the support for smooth transitions during zoom and illumination changes. We have generalized the levels in the vertical dimension of a TAM to integrate with results from traditional mipmaps to allow customization based on spatial coherence requirements. Our TAM implementation also supports the changing of hatch styles such as 90-degree or 45-degree cross hatching.

The Multi-Texture Blending shader reproduced the results from Praun et al. in real time. Our rendered results present objects with seamless hatch strokes and appear natural and resemble those of hand-drawn hatch artwork. Our implementation integrated and supported interactive manipulation of effects from general illumination models including specularity, light source types, variable hatch and object colors, and rendering of surface textures as cross hatch. Additionally, we investigated trade-offs between per-vertex and per-fragment tone computation and discovered that the smoothness in hatching can be better captured in the per-vertex

computation with the lower sampling rate and interpolations. Finally, the novel integration of TAMs and traditional mipmaps allow customizable spatial coherence support which allows smooth hatch strokes and texture transitions in animations during object size and illumination changes.

TABLE OF CONTENTS

ACKNOWLEDGEMENTS

I would like to personally thank my committee chair, Kelvin Sung, for his patience, guidance, supportive advice, and caring not only throughout this thesis process but also for my wholistic career development. I would also like to thank my other committee members, Michael Stiber and Yusuf Pisan, your questions and feedback have made this thesis more complete and reader friendly. Next, I would like to thank my parents, who always supported me both mentally and financially along this journey. Finally, I would like to thank my husband, Fengquan, for always understanding and providing support in busy late nights and stressful school days.

Chapter 1. INTRODUCTION

1.1 NON-PHOTOREALISTIC RENDERING

Non-Photorealistic Rendering (NPR) is an important research area in computer graphics. NPR refers to the type of rendering technique on the opposite side of photorealistic rendering. Photorealistic rendering focuses on reproducing the appearance of the real world. On the other hand, NPR focuses on simulating a particular artistic style that does not necessarily reflect the real world [1]. Figure 1.1 shows the difference between photorealistic rendering and NPR, with the photorealistic picture on the left almost looking like a real-life photograph and the NPR picture on the right showing off a house with watercolor painting artistic style.

Figure 1.1 Photorealistic rendering vs. NPR [2] [3]

1.2 WHY CHOOSE NPR

NPR has always been the interest of researchers and practitioners. There is a conference dedicated specifically to NPR: “NPAR: Non-Photorealistic Animation and Rendering” [4]. Different types of NPR have also been used in a wide variety of interactive systems such as games. For example, “Okami” from Capcom (Figure 1.2a) used water-color painting NPR, and “The Legend of Zelda: The Wind Waker” (Figure 1.2b) from Nintendo used toon shading.

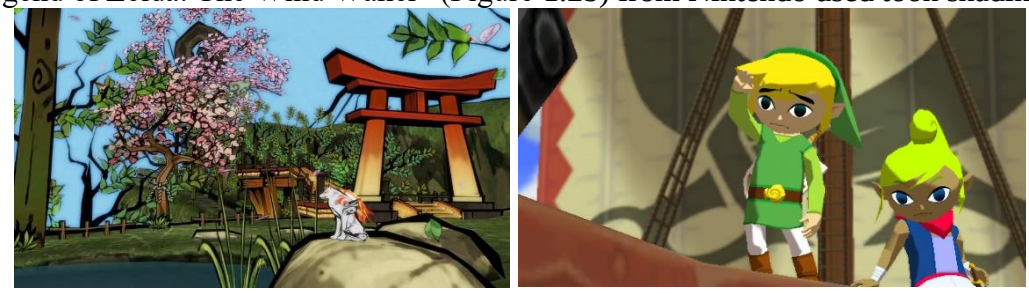


Figure 1.2 a) "Okami" from Capcom; b) “The Legend of Zelda: The Wind Waker” from Nintendo

Hatch rendering is an example of NPR that is unique and interesting. The main goal of NPR hatch rendering is to reproduce the hand-drawn hatching artistic effect in computer graphics.

1.3 What Is Hatch And Why Choose Hatch

Hatching is one of the most commonly used and popular techniques in drawing for centuries [5]. Hatching as a drawing technique is commonly used in different types of drawings such as pen-and-ink and graphite. It is also an important part of traditional printmaking such as engraving [6].

There exist multiple hatch techniques. Some of the basic hatching techniques are parallel hatching, contour hatching, and cross hatching. This project aims to reproduce the third basic type of hatching techniques: contour cross hatching. Figure 1.3 shows the three examples of common hatching techniques. Parallel hatching places parallel straight lines closely together to convey the object's volume, shape, and illumination. In contour hatching, strokes follow the contour of the object which further increases the sense of volume and shape. Contour cross hatching uses crossing lines instead of only parallel lines to inject a richer sense of the object's volume, shape, and illumination. In all the techniques, higher density of strokes represents darker places on the object. All these techniques can be combined to deliver aesthetic hatch art [7].

Figure 1.3 Parallel, Contour, and Cross Hatching [7]

Hatching is a unique artistic style compared to other commonly seen techniques such as watercolor painting and oil painting [8]. This brings unique challenges to hatch rendering in real-time interactive systems. Many other techniques, for example, such as watercolor as illustrated in Figure 1.4, where contour and illumination of the hand can be captured with appropriate colors. However, pen-and-ink hatching hardly conveys any tone or color with a single stroke itself. Hatching can only use multiple strokes' combined qualities such as their density, length, and orientation to express the object's shape, volume, and contour [8]. This presents unique challenges when reproducing hatch artistic style in NPR.

Figure 1.4 Watercolor painting [9]

Moreover, other types of NPR are commonly seen in interactive systems such as games and animations, hatch style NPR is less frequently encountered. This is interesting because hatching is a common and popular art technique just like other NPR styles such as watercolor painting. In fact, the hatch art style being used in games is mostly 2D or only non-real-time animations.

Therefore, it is worth investigating and exploring hatch shading in these types of interactive systems.

1.4 Design, Implementation, and Results Overview

This project strives to study and reproduce an interactive contour cross hatching 3D rendering system. The project adopts the texture based and Object space approach Praun et al. [10] developed and the solution Lele Feng [11] demonstrated. Design details and the success evaluation criteria are discussed in Chapter 3.

The project was implemented in Unity game engine. Unity is a well-developed and well-maintained platform that provides all the necessary tools to achieve our project goals. Detailed technical tools justification is discussed in Chapter 3. Chapter 4 will go into the details of the implementation process.

We achieved the goal of hatch rendering 3D objects in real time with user-controlled parameters demonstrating potentials of hatch rendering being customizable and generalizable. The hatch rendering system supports hatch tone computation related to observed artistic styles. We further generalized the results to more complete shading models, including specularity, multiple light sources, different types of light sources, colored light sources, colored hatch, different hatch styles, and object surface textures. We also achieved the support of potentially contradictory spatial coherence conditions by unifying TAM and traditional mipmaps. We also demonstrated support for animations. Lastly, we investigated the tradeoffs of per-vertex and per-fragment illumination computation and identified the suitable use cases for each: animation for the smoother results of per-vertex and static images for the sharper results when object size is large from per-fragment computation. Chapter 5 will discuss in detail about all the results. Figure 1.5 is a quick preview of the final results from this project.

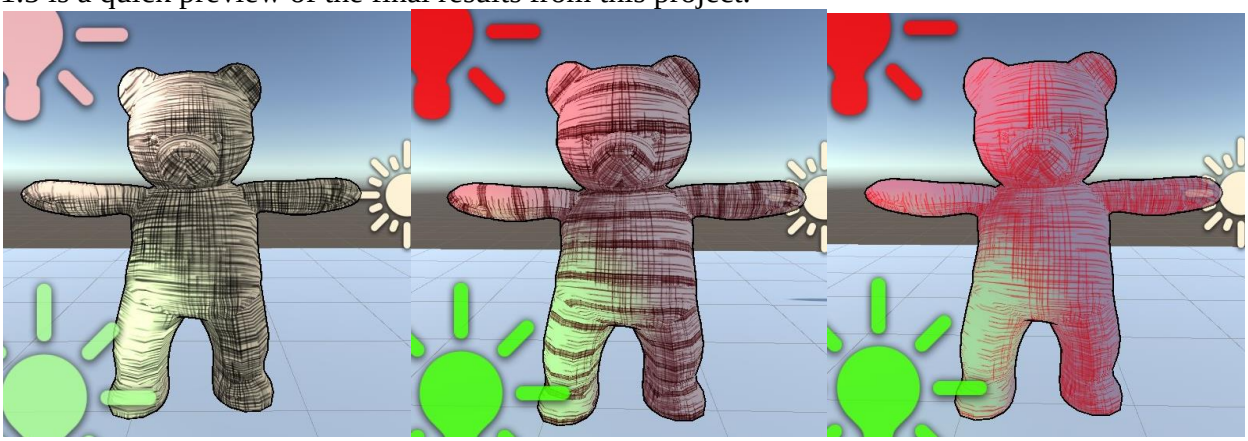


Figure 1.5 A preview of the final results of this project on illumination, colored hatch, and underlying texture.

Chapter 2. RELATED WORKS

Existing solutions to hatch rendering can be generally classified into two approaches: primitive-based and texture-based [12]. Primitive-based approaches generate new geometries representing hatch strokes while texture-based approaches mimic the artistic style of hatching via applying pre-computed textures. The solutions to both of these approaches can be derived in either image space or object space [13]. Image-space solutions apply the geometry strokes or the texture in screen space after 3D objects are rendered into pixels. On the other hand, object-space solutions apply the geometric strokes or the texture onto the surface of the 3D objects.

It is also worthwhile to define mipmap before we discuss in detail. Mipmap is a commonly used technique to improve rendering speed and reducing aliasing effects [14]. A mipmap is a sequence of pre-computed textures of the same image, with progressively lower resolution. During runtime, texture resolution to be sampled is selected according to projected object size to avoid aliasing [15].

2.1 PRIMITIVE- AND TEXTURE-BASED SOLUTION

Primitive-based approaches need to generate new geometry for each stroke. This allows primitive-based methods to have a finer control of the properties of every hatch stroke, such as length and placement. However, this approach must also compute the orientation and the placement of every single stroke in the scene which requires significant computation. For instance, Umenhoffer et al.'s primitive-based and screen space approach first generated all the strokes in screen space, then they rejected strokes in overpopulated areas, filled in underpopulated areas, and oriented each stroke based on illumination [13]. For this reason, primitive-based approaches have high computation costs which can impair the performance of real-time rendering.

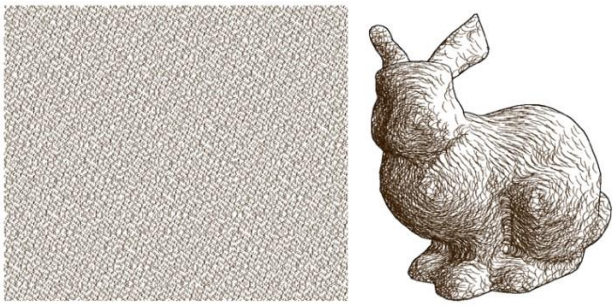


Figure 2.1 Hatch strokes initially defined in screen space, then applied to the bunny object from Umenhoffer et al. [13]

On the other hand, texture-based methods usually have a better real-time performance when compared to primitive-based methods since the textures are usually pre-computed. Additionally, texture-based methods can usually generate the hatch texture from a hand-drawn stroke which enables the final hatch rendering to appear more similar to hand-drawn hatch art. Being able to generate texture from hand-drawn hatch strokes also allows users to control finer details of the strokes for a wider variety of natural-looking hatch art. Though primitive-based methods might have a finer control of each stroke's placement and orientation which helps depict the object's characteristics, primitive-based methods usually lack the hand-drawn effect since every stroke

has the same perfect machine-generated appearance. For these reasons, texture-based methods are a more applicable approach for real-time hatch rendering. Texture-based example work will be discussed in detail in the next section.

2.2 IMAGE AND OBJECT SPACE SOLUTION

Image-space solutions usually have better spatial coherence. Since the strokes are applied to the screen space, the stroke density and width in the whole scene can be kept uniform when zooming in and out the scene. This makes the entire scene always look coherent as it is from a single piece of hatch art. However, image space solutions usually suffer from the "shower door effect". This is an effect where the viewer has a sense of looking at the scene through a semi-transparent layer in which the strokes or textures are embedded. This can significantly defeat the hatching style that the viewers expect in a real-time interactive system. Moreover, since strokes are applied in the screen space, it can be challenging to align the strokes with the object's contour, making contour hatching harder to achieve.

For example, in their primitive-based and image-space approach, Umenhoffer et al. followed the particle movements according to the screen-space velocity field to place the screen-space strokes. After which, they developed a rejection-based algorithm to filter for regions requiring more or fewer stroke placements, allowing the density of strokes to correspond with the illumination condition [13]. This process is depicted in Figure 2.2. This method has the advantages of uniform stroke density and width in screen space. However, as discussed, the computation requirement is significant and unrealistic for real-time applications.

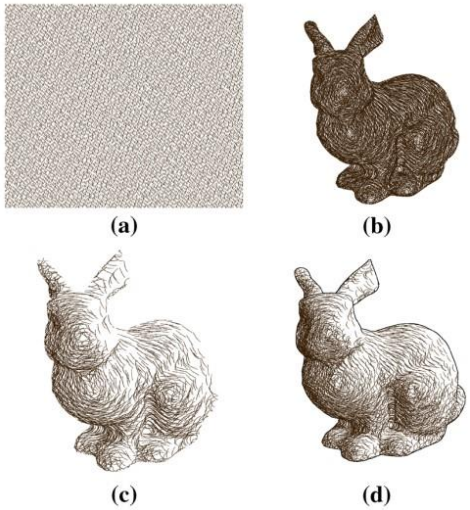


Figure 2.2 Stages of the stroke rendering algorithm from Umenhoffer et al.: a) hatch strokes placed in the screen space; b) stroke orientation process; c) Based on illumination, strokes are rejected; d) edge detected contours are added [13]

In contrast to screen space solution, object space solution has the advantage of temporal motion coherence. Because the strokes are applied directly to the surface of the object, the strokes can move with the object. In this way, the "shower door effect" can be avoided. Additionally, since the stroke textures are applied directly onto the object surface, the orientation of the strokes can easily follow the contour of the object, making contour hatching easier to achieve. The challenge of object-space solution is the uniformity of stroke density and width in the entire scene. Figure 2.3 demonstrates a hatch art where all the strokes used in the scene have

approximately uniform width and density. For example, the far wall of Figure 2.3 and the grass at the front has roughly the same width and the same density.

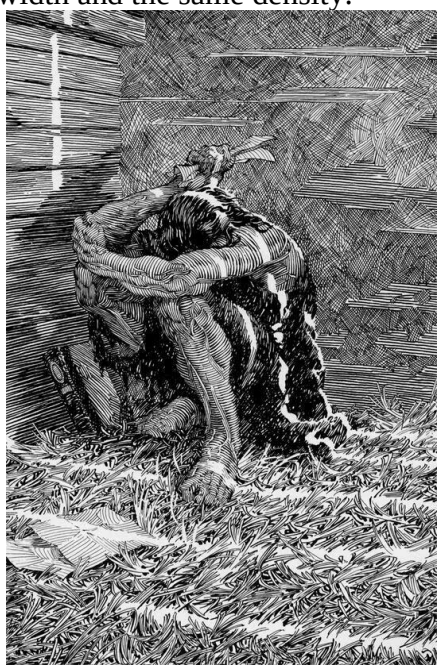


Figure 2.3 A hatch art showing roughly uniform stroke width and density throughout the entire scene [16]

However, this is not the only type of popular hatch style. Some hatch arts do use slightly thinner and shorter strokes for smaller and further objects if they are too small or too further away in the scene. As Figure 2.4 shows, people in the bottom left corner or the cloud in the sky are with thinner and shorter strokes. Whereas the main subject standing in the front is with longer and thinker strokes. In general, shorter strokes can be used for smaller objects, thinner strokes can be used for further and smaller objects.

Figure 2.4 One hatch art using shorter and thinner strokes on smaller and further objects [17]

This different spatial requirement challenge can be addressed. For example, as illustrated in Figure 2.5, Praun et al. addressed this problem by introducing Tonal Art Maps (TAM) and custom mipmapping [10]. The final results show reasonable spatial coherence across screen space. TAM is a matrix of textures pre-computed to capture projected object size changes on the vertical dimension and illumination changes on the horizontal dimension. The vertical dimension of TAM serves as a customized mipmap where all strokes in the lower resolution texture are present in the higher resolution textures. Similarly, on the horizontal dimension, all strokes in the lighter texture are present in the larger and darker texture. As illustrated in Figure 2.5, the textures along each vertical column are mipmaps meant to ensure spatial coherency, and the textures along each horizontal row are meant to capture illumination differences and ensure a smooth transition during illumination change. Notice that textures in both the vertical and horizontal directions are with strokes that share the same width, and the corresponding strokes must maintain the corresponding position and length. This nesting property along the vertical columns allows existing strokes to be switched to their shorter version or longer version when zoomed in or out. Where in the horizontal rows, this nesting property ensures more strokes will appear or disappear, when illumination condition changes.

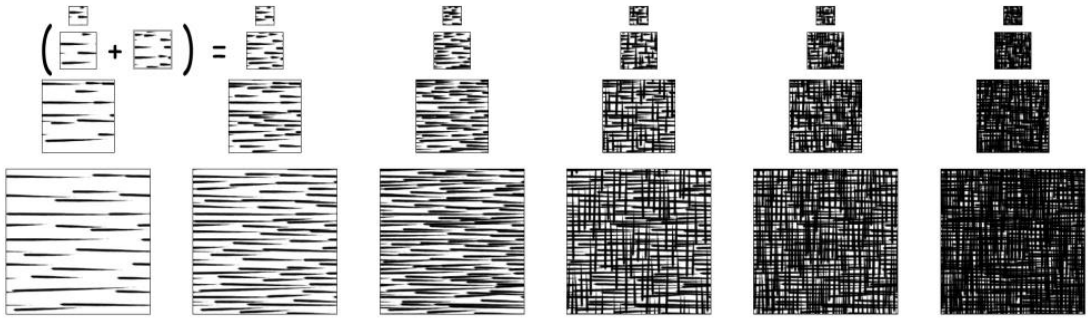


Figure 2.5 Tonal Art Map introduced by Praun et al. [10] including all the textures used

As discussed above, texture-based methods are usually with more efficient computation and better run time performance. They also benefit from the hand-drawn hatch strokes, which can result in the rendering seemingly more natural and aesthetically pleasing. Although these methods have varying spatial requirements, the challenge can be successfully addressed by the TAM method Praun et al. [10] proposed. Praun et al. presented a texture-based and object-space method that was based on conventional texture sampling and multi-texture blending [10]. Multi-texture blending is a technique where at any position on the object surface, blended results of textures are used. In this project, we followed Lele Feng's implementation [11], where a blended result of two adjacent-toned textures (along horizontal dimension) based on hatch lookup value is calculated to be the final hatch texture on a particular position (vertex/fragment). We use linear blending where the distance from the lighter tone value to the hatch lookup value is the blending weight of the darker tone texture; and the distance from the hatch lookup value to the darker tone value is the blending weight of the lighter tone texture. Figure 2.6 shows an example of multi-texture blending where the resulting texture on the right is an equal blend from textures A and B.

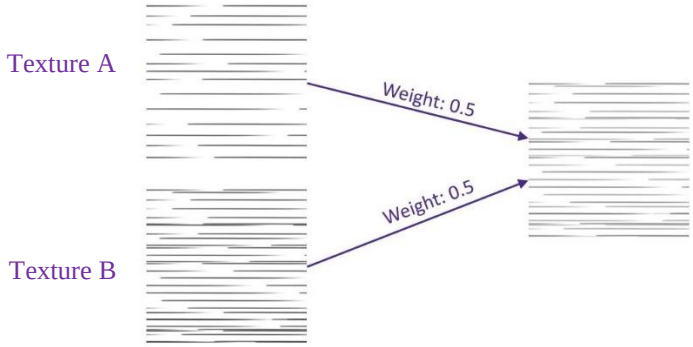


Figure 2.6 Example of multi-texture blending process.

The TAM generated and used in their method are depicted in Figure 2.5. Figure 2.7 shows some of the real-time rendering results. As shown in Figure 2.7, this texture-based and object-space solution demonstrates natural hatch rendering results that are aesthetically close to hand-drawn hatch art. Because of its object space characteristic, the rendered 3D objects have great temporal coherence. In addition, an appropriate TAM structure can also deliver reasonable spatial coherence. Because this approach presents a relatively complete solution where the results are aesthetically pleasing, this is the approach that we decided to study and reproduce.

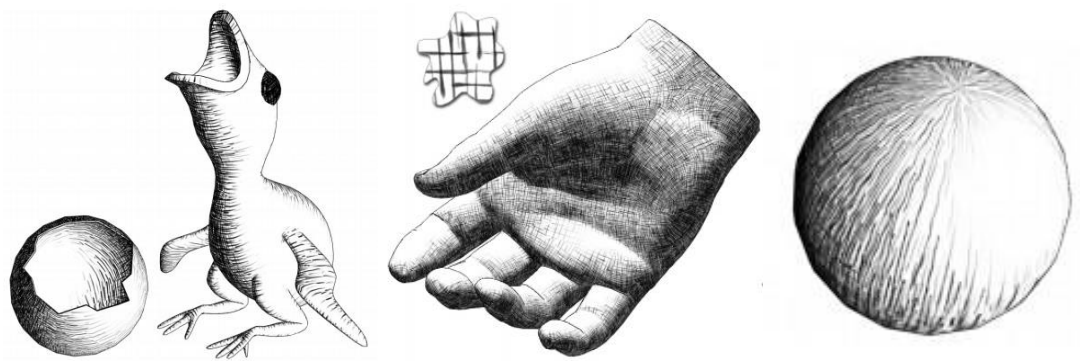


Figure 2.7 Some rendering results produced by Praun et al. [10]

2.3 PROJECT GOALS

Our background research led us to choose to reproduce 3D hatch rendering in real time with the texture-based and object-space Praun et al. solution [10].

The two major components of achieving this goal are TAM generation and illumination computation based on multi-texture blending. TAM is important to support spatial coherence requirement. When users zoom in and out, and when a scene contains both large and small objects, proper deployment of TAM can ensure a uniform stroke width and density. This keeps the entire scene appearing cohesive and belonging to a single hatch art. Considering the requirements presented in Figures 2.3 and 2.4, where the variability of stroke density and size is tied to artistic expression, this project will also examine different customization approaches to creating TAM.

The other major component of the project goal is to simulate illumination based on multi-texture blending. We adopted the Praun et al. [10] approach and referred to the implementation of Lele Feng [11] with the aim to expand the results to include more generalizable use cases, such as different illumination models, illumination from multiple light sources, different types of light sources, colored hatch, and hatching of the underlying textures on the object. We also aim to provide an extensive graphical user interface that allows the user to customize the final hatch rendering results based on their needs.

Chapter 3. SPECIFICATION AND DESIGN

3.1 SPECIFICATION

In this project, we will reproduce and improve upon 3D hatch rendering in a real-time interactive system with the texture-based solutions in object space presented by Praun et al. [10] while referring to the solution Lele Feng demonstrated in her book [11]. Here are the specifications of our system:

1. Flexibility: Users should have the option to manipulate multiple parameters to achieve different hatch rendering results, such as different hatch styles, different colors of hatch, underlying textures, object color, and colored lights.
2. Generalizability: The rendering process should support a general shading model, such as variable number of light sources, light source characteristics such as types and colors and diffused and specular illumination.
3. Performance: The 3D object hatch rendering and parameter manipulation should be in real time.
4. Quality of Rendering: the effect of hatch rendering should look aesthetically pleasing and resemble hand drawn hatch art.
5. Animation: animations should be supported on the hatch rendered 3D objects without texture swimming.

This specification implies that the target system must deliver the following functionality:

1. Texture mapping on objects based on the defined UV coordinate
2. Appropriate TAM to support spatial coherence, smooth transition during illumination and zoom changes, and uniform distribution of strokes required for hatch art
3. Appropriate TAM generation process that supports various hatch styles
4. Multi-texture blending technique to support smooth transition of strokes fading effect during illumination and projected object size change

Additionally, we also aim to further generalize the results by supporting:

1. A general illumination model including support for diffuse and specular effects
2. Multiple light sources with appropriate parameters include light source colors
3. Generalization of hatch art including: colored hatch, hatching of underlying texture on objects, and colorization of hatch art

3.2 SOLUTION DESIGN

As discussed, the solution Praun et al. [10] presented is an object-space solution and possess the inherit advantage of straightforward hatching following object contours and temporal coherence. By appropriately defining texture coordinate and applying hatch texture based on the resulting UV on the object, the stroke orientation naturally follows the object surface curvature, and the texture follows the movement of the object movement, resulting in great temporal coherence.

3.2.1 *Illumination and Cross Hatching Solution*

Stroke density in hatch art represents illumination conditions. As shown in Figure 3.1, the density of strokes conveys the sense of tone, where lower density conveys a lighter tone and higher density results in darker tones. And this tone is defined as percentage of pixels that is covered by the strokes (percentage of pixels that are nonwhite). This requires a set of textures with a range of stroke densities to reflect the corresponding light to dark tones. Because in our TAM texture sizes are constant, more strokes in a texture means higher density. Therefore, as shown in Figure 3.1, increasing number of strokes are added from column 1 to column 16 to represent light to dark tones.

For cross hatching arts, artists gradually shift from low density parallel hatching to high density parallel hatching, and finally to cross hatching as illumination changes from light to dark. To capture this property, we first continuously add horizontal strokes to the textures from column 1 to column 7 (tone 0.172 – tone 0.912). Then from column 8 to column 11(tone 0.912 – tone 0.976), we only add vertical strokes. Finally, from column 12 to column 16 (tone 0.984 – tone 1.0), we add either horizontal or vertical strokes on a fifty percent chance. This stroke adding logic is adopted from Praun et al. [10] and when to start adding strokes at another direction for cross hatching can be configured by the user.

Lastly, the artistic style of the hatch texture can be customized. For example, the direction of the strokes can be customized. In this project, we investigated 90°, 45°, and 25° cross hatching. Different degrees in the name indicates the angle of the second layer of strokes. Different angle of the second layer of strokes are different artistic options. Figure 3.1 shows 90° cross hatching. Other sets of TAMs will be shown and discussed in implementation section. The tone represented by each column is also customizable by users.

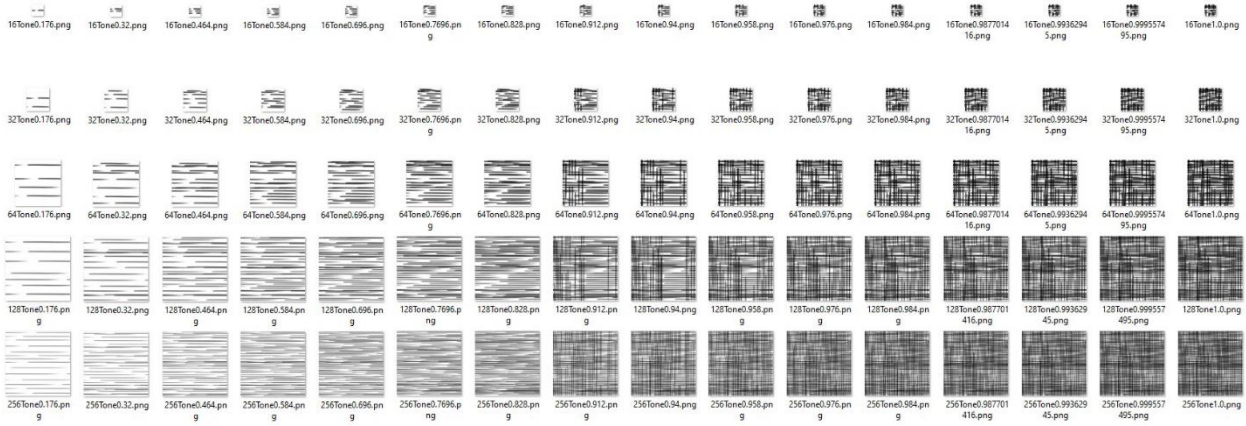


Figure 3.1 90 ° cross hatching TAM generated and used in this project with 16 tone columns, and gamma correction from Praun et al [10]

3.2.2 *Consideration for a Hatch Texture*

Hatch arts requires strokes be placed in approximated **uniformity**. In the context of hatch rendering, this requirement means uniformity of strokes **across the entire texture** because all spaces with the same density represents the same tone. For this reason, additional strokes for increasing tones must be included in the texture strategically ensuring the distribution uniformity.

We adopted stroke quality measurement recommendation from Praun et al. [10] and Liatis [18, p. 74]. The quality of a stroke is measured by two indexes: **Literal Tone Difference**, and **Gaussian Pyramid Difference**. Literal Tone Difference measures the literal tone contribution of the current stroke to the texture, which is the percentage of nonwhite pixels that the new stroke would contribute to the texture. It measures the level of overlap of the new stroke when it is applied to the texture. Each hatch stroke should be placed so that it has as less overlap as possible, meaning it should have tone contribution to the texture as large as possible. Gaussian Pyramid Difference measures the uniformity of distribution when the current stroke is added to the texture. A Gaussian pyramid is created by recursively applying gaussian filter then reduce the resolution by a set number of times. This index is the total sum of the tone differences between textures with and without the new stroke at each gaussian pyramid level. The larger the index is, the more tone contribution even after gaussian blur and shrinking is, meaning less overlapping or too closely drawn strokes. The combination of the two indexes defines whether this current stroke is the best fitted stroke.

Additionally, because the object sizes in a scene can vary, the support of tiled texture is also important. This requires each mipmap texture to be toroidal, which means stroke continuity must be maintained across texture boundaries.

3.2.3 *Screen Space Spatial Coherence Solution*

As discussed in the related works section, maintaining spatial coherence over the entire screen space is one of the main challenges of object space solutions. For example, when copies of the same object are at different viewing distances with very different sizes, without special attention, the hatch strokes on these objects will be distinct in both width and density. For example, Figure 3.2 shows the same copy of bear appears at three distinct sizes and viewing distances. These bears are defined with the identical stroke texture. The result of traditional mipmap rendering shows that the texture, or stroke size, is scaled proportionally. This designed results from ordinary mipmap, while is expected in a more traditional graphics renderings, in the case of hatch rendering, such variation of stroke characteristic may result in an inhomogeneous composition and be perceived as being distinct from a hand drawn hatch artwork. For a typical hand drawn hatch art, one would expect similar stroke widths on all the objects in the same scene.

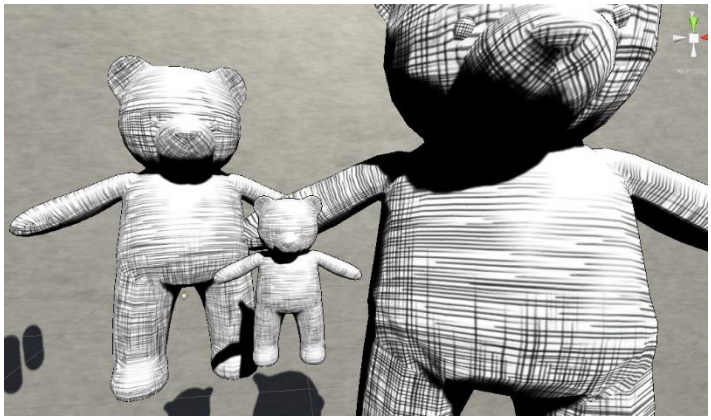


Figure 3.2 Bear models with different sizes and distances to the camera with normal mipmap

Because the traditional mipmap is not designed to address this unique spatial coherence requirement, Tonal Art Map (TAM) was introduced by Praun et al. to address this problem [10].

The unique structure of TAM is designed to address the spatial coherence challenges. As Figure 3.3a shows, in ordinary mipmaps lower resolution textures are filtered from the original texture by averaging corresponding pixels. This results in the stroke width to be thinner and with a higher stroke density in lower resolution textures. This drastic changes in the width and density of strokes, as illustrated in Figure 3.2, can result in lack of homogeneity in the composition—it is as though the three bears are distinct from each other. In contrast, Figure 3.3b shows that the TAM maintains the same stroke width and stroke density, which ensures the spatial coherence and the tone in a single tonal column.

a b

Figure 3.3 a) Ordinary mipmap of a hatch texture; b) TAM's mipmap column of the same hatch texture

3.2.4 *Smooth Transition During Object Movement and Illumination Change*

When the projected size of an object changes over time, to avoid texture swimming, mipmap should react by selecting an appropriate texture resolution for sampling. For example, referring to Figure 3.3b, in a closeup when the projected size of the object is large, a higher resolution texture (such as the bottom texture of a TAM) will be chosen, while when the object is moved far away from the camera and with a smaller projected size, the texture with lower resolution (the top texture in the same column of a TAM) will be chosen.

As Figure 3.4b demonstrates, the strokes that appear in the lower resolution texture (higher up in the column of a TAM) are also present in the higher resolution texture. This is illustrated by the blue arrows pointing to the corresponding strokes in different textures. The position and length relative to the texture size that the stroke resides are maintained. This ensures smooth transition when mipmap levels switch at run time and gives users the impression of the same stroke getting longer or shorter without width change.

When illumination conditions change, the choice of texture should also be changed to reflect the different tones. This change of hatch texture resulting from illumination tone changes should also be smooth. The smoothness in tradition requires no sudden changes in texture strokes which dictates that the strokes exist in the lighter tone textures must be present in the dark tone textures. For this reason, darker tone textures will be synthesized by adding new strokes to the existing lighter tone textures. In this way, switching between light and dark tone is equivalent to including and removing strokes on the current texture, without sudden changes to a completely different texture. As shown in Figure 3.4, corresponding colored arrows point at the same strokes

in different textures. Exact strokes are kept in the same mipmap levels of all columns, where the textures in the same row are with additional strokes conveying a darker tone.

a b

Figure 3.4 a) TAM column of hatch texture of tone 0.172; b) TAM column of the next column hatch texture of tone 0.32

Furthermore, to support smooth appearing and disappearing of strokes, the multi-texture blending technique is adopted from Praun et al.[10] and Lele Feng [11]. We blend two adjacent-toned textures with weights corresponding to illumination condition. For example, brighter illumination corresponds to more weights on lighter texture and less weights on darker texture. Therefore, when illumination gradually changes, the texture weights changes accordingly, which means strokes gradually fade in or out instead of sudden appear or disappear.

It is interesting that the sense of homogeneity of hatch art composition can potentially result in **contradictory spatial coherence requirements**. As highlighted in Figure 2.3, the requirement of approximate stroke density and width throughout the entire scene vs. as discussed in Figure 2.4, artistic preference of relatively shorter and thinner strokes used on objects with very small projected size. TAM addresses the first spatial requirements, whereas it does not address the second artistic preferences. In order to unify these two seemingly contradictory spatial coherence requirements, we need to **integrate TAM and traditional mipmap**. In this way, TAMs can be used on objects with large to medium projected sizes. When the projected size is sufficiently small, the traditional mipmaps can be invoked to fulfill the expectation to use relatively shorter and thinner strokes.

3.2.5 *Generalizations*

Besides simple illumination situations supported by Praun et al. [10] and Lele Feng [11], we also aim to support more general illumination models. Therefore, we include specularity, which requires one more specular illumination computation besides diffuse illumination. We designed to provide the option for user to choose between displaying only diffuse illumination, only specular illumination, or both. Additionally, we also aim to support multiple light sources, which requires each light's illumination condition been computed on every vertex or fragment. We designed to provide users the options to display the maximum contribution among the light sources or the addition of all the light sources. Addition option provides the most alignment with the real-world physics and maximum option provides more rendering details of every light

sources. Light colors are also considered into the final rendering results for novel hatch colorization investigation.

Object colors are not considered by Praun et al. [10] and Lele Feng [11]. However, object colors can be an important consideration in hatch art. As Figure 3.5 shows, while the bottle and the pitcher on the right are in close proximity and under similar illumination condition, the former has a much darker hatch texture. The darker texture conveyed a sense of darker color of the bottle when compared with that of the pitcher. The specular highlights on these objects also serve as examples of incorporating specular illumination in hatch art. In order to support this feature, we need to consider object color into hatch lookup value. Additionally, we also provide the potion to show object color for novel hatch colorization investigation.

Figure 3.5 Example of object color considered into hatching texture selection in hand drawn hatch art. Giorgio Morandi etching. Still Life with Five Objects, 1954 [5]

Besides hatch rendering to express illumination and volume, **underlying texture** is also an important component of some hatch arts. There are two types of underlying texture: not hatched underlying texture; and hatched underlying texture. Figure 3.6 and 3.7 shows two types of texture hatching in hatch art. Figure 3.6 shows hatch over underlying textures (floral pattern on the lady's dress) where the underlying textures are not hatched. This approach is usually used for underlying textures that are not suitable to be represented by hatch, such as complicated patterns like flowers [19]. Figure 3.7 shows an example of hatch over underlying textures (stripe patterns on the lady's dress) where underlying textures are also hatched. This approach is usually used for underlying textures that are suitable to represent by hatch, such as simple stripes and checkers [20].

Figure 3.6 Example of hatch over underlying textures (floral pattern on the lady's dress) [19]

Figure 3.7 Example of hatched underlying textures (stripe pattern on the lady's dress) [20]

We aim to support both types of underlying textures in our project. Because the underlying texture is a separate process independent from the illumination process, occurring at the end of the rendering process.

3.3 Implementation Platforms and Tools

TAM is a crucial component of our project. TAM can be drawn by hand as Praun et al mentioned [10]. However, the accuracy of calculating each stroke's relative position and length in larger textures and the placement of the longer strokes can be challenging. Therefore, an automated TAM generator is the most efficient approach. In Praun et al.'s approach, TAM generation is performed once with results being reused in all scenes [10].

For this reason, TAM can be implemented as a separated system on the CPU side. TAM generation requires a system that can take a stroke image as an input and output a texture image with additional strokes. It also needs to support manipulation and placement of the stroke image. Lastly, this system must support a fast feedback loop. Quick and easy preview of the generated test textures or manipulated strokes can greatly improve the exploration of the TAM generation in the beginning of the project.

After TAM textures are computed, it is required to construct each TAM texture column into mipmap enabled file type used by Unity. We also would like the flexibility to switch back to ordinary mipmap at any mipmap level in our TAM texture column. To achieve these goals, we need some graphics tool for mipmap construction and edition.

In order to render 3D objects in hatch style, the system implementation requires operations both on the CPU and GPU sides. The CPU side operation should support the manipulation of 3D objects and the user interface for the creation and manipulation of scene components such as light sources and the associated properties. The GPU side operation should support the illumination computation and the texture lookup process.

- **Jupyter Notebook:** Based on the described TAM generation requirements, a system with fast feedback loop is required for us to preview the manipulated strokes and generated textures. Jupyter Notebook is a well-suited platform since it satisfies all the above requirements. Jupyter Notebook is a web-based light-weight application that allows the user to compile and run the code for all aspects of a data project in one place. This allows us to easily visualize data of the generated TAM and the generation process, if necessary, with very minimal setup and configuration. In addition, the code can be run and tested in a modular fashion, which makes the TAM generation trial and error process easier to observe, experiment and manage.
- **PVRTexTool:** Based on the described requirements, an image processing tool for mipmap chain construction and mipmap enabled file type encoding for Unity (Direct Draw Surface) is required. PVRTexTool supports all the above requirements.
- **GIMP:** Based on the described requirements, an image processing tool for mipmap edition at each resolution level is required. GIMP is chosen because it satisfies all the above requirements.
- **Unity:** Based on the described requirements, solutions implementation requires access to both CPU and GPU side operations. In addition, the solution also requires convenient 3D model import, and manipulation. Moreover, a convenient approach to expose parameters is required to better support user customization of the rendering process. Unity is a popular cross-platform game engine that satisfies all the above requirements. Because of the previous course works and practices, we have an experience and foundation knowledge of Unity. Therefore, we chose Unity for our implementation.

Unity requires minimal configurations and setup, and also provides easy to manipulate user interfaces. This allows users to experience different parameter settings at run time which greatly supports our investigation and implementation of our real-time 3D object hatch rendering system. Moreover, because of the well-developed community for Unity, various third-party 3D objects and animations can be readily accessed. The integration and manipulation of the 3D objects and animations are also well-supported which greatly helps the project investigation and results demonstration.

Chapter 4. TAM IMPLEMENTATION

Figure 4.1 shows the overview of our solution design. (Process: TAM generation, Scene setup, C# script and shader construction.)

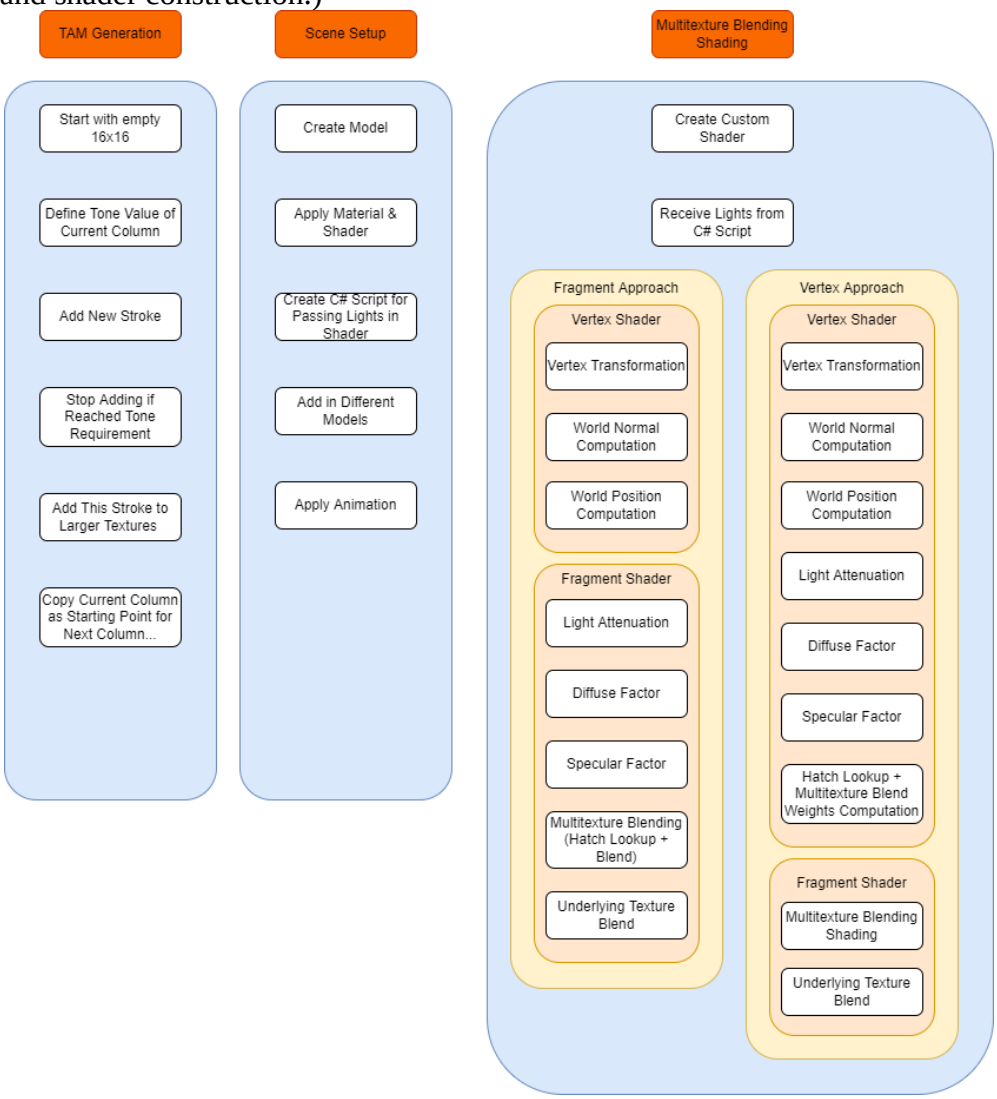


Figure 4.1 Solution design overview diagram

As discussed in previous chapters, Tonal Art Map (TAM) is a crucial part of this project. According to Praun et al. [10], TAM can also be drawn by hand. However, in this project, the generation of TAM is fully automated. The current TAM generator is written in python and supports parallel-hatching texture and cross-hatching texture. The approach of TAM generation in this project followed Praun et al.'s approach [10] and referenced Liatis's implementation [18]. TAM is generated from the lightest tone to the darkest tone (left to right of Figure 2.5), and from the lowest to the highest texture resolution (top to bottom of Figure 2.5). We start from the lowest resolution texture in the first tone column, then we proceed to the higher resolution texture in the same column. After generating the current column, we proceed to the next darker column to the right. Each texture of the darker column is created by adding new strokes to the corresponding texture of the lighter tone column to its left.

4.1 TAM HIERARCHY

4.1.1 *TAM Single Column Generation Process*

As discussed in chapter 3, in order to achieve both spatial coherence and temporal coherence, we need to construct the hierarchical structure of TAM such that:

1. Strokes present in lower resolution textures must remain in higher resolution ones.
2. Strokes present in lighter-tone must remain in darker-tone ones.

Once added, all strokes must remain in the same position and length relative to their current texture size. The TAM generation method that Praun et al. [10] presented was to add strokes from the small textures to large textures (top to bottom) and from the light tone texture column to the dark tone texture column (left to right). Every stroke to be added is selected from multiple candidate strokes, and each candidate is generated with a random length and placed at a random position. Because artists prefer longer strokes in hatching, the stroke length was set to at least 40% of the texture resolution. Candidate strokes are evaluated by two indexes: **Literal Tone Difference**, and **Gaussian Pyramid Difference**.

Each candidate stroke is evaluated by these two indexes and the one with the best combined index value is kept as the best-fitting stroke. Each time a best-fitting stroke is selected, this stroke is also applied to all the higher resolution textures in the same column. The strokes in the higher resolution textures maintain the same width and position relative to the texture size. The stroke adding process continues until the current texture has reached a threshold of required tone, current setting is within 2% of the target.

As discussed in Chapter 3, in order to map the TAM textures to 3D objects with different-sizes, **tileable textures** are required. For this reason, stroke continuity across tiled boundaries must be maintained. After applying a candidate stroke to the current texture, we record its position in the texture and evaluate whether the stroke is truncated and when required, perform the necessary wrapping operation on all four boundaries.

Finally, this project demonstrates the **generalization of different cross-hatching styles** by showing 90°, 45°, and 25° cross hatching, where the second layer of strokes are rotated at a 90°, 45°, and 25° angles. Praun et al. [10] also showed that this TAM generation process is generalizable to fit other hatching styles such as stippling.

4.1.2 *TAM Columns Generation*

This process of computing for a column is repeated for each subsequent columns to the right. Recall that columns to the right are with darker tone and must retain all strokes that exists in the previous column. For this reason, new columns are computed by making a copy of the current column as the starting point for adding new strokes. In this way, tone changes are accomplished by sampling from different columns of the TAM with strokes added or removed and thus a smooth transition between different illumination condition can be accomplished.

4.2 BEST STROKE SELECTION FROM CANDIDATES

Candidate strokes are evaluated by two indexes: **Literal Tone Difference**, and **Gaussian Pyramid Difference**. Each candidate stroke is evaluated by these two indexes and the one with

the best sum index value is kept as the best-fitting stroke. Literal Tone Difference measures the tone progress the new candidate stroke brings to this current texture. Gaussian Pyramid Difference measures the distribution uniformity of the new candidate stroke when applied to the texture.

4.2.1 *Literal Tone Difference*

Literal Tone Difference is calculated as the difference between the texture with and without the new stroke. The result represents the tone contribution that the new stroke brings to the texture. Since hatch arts require an even distribution of the strokes with minimum overlap, the preference is the stroke with largest tone contributions.

4.2.2 *Gaussian Pyramid Difference*

Gaussian Pyramid Difference is calculated as the sum of the difference between each level of the Gaussian Pyramid with and without the new stroke. Gaussian Pyramid is created by applying the Gaussian filter to blur the current texture and then reducing the resolution by half. To avoid over domination from lower resolutions, in this project, the filtering process ends at 4x4. we chose to perform 2 times for 16x16 texture, 3 times for 32x32 texture, 4 times for 64x64, 5 times for 128x128, and 6 times for 256x256 texture. After creating the Gaussian Pyramid, at each level, we find the tone difference between the Gaussian texture with and without the new stroke. The sum of the tone differences is the final Gaussian Pyramid Difference. This value indicates the evenness of distribution with and without the current stroke. This is because the above Gaussian Blur processing would merge a new stroke that is placed too close to an existing stroke. For this reason, strokes that are in close proximity will result in a smaller Gaussian Pyramid Difference. Because hatch arts require an even distribution of strokes across the texture, the preference is a stroke with the maximum Gaussian Pyramid Difference.

4.2.3 *Best Stroke Selection Discussion*

It is important to recognize that it is challenging to define what is the best stroke as in general, the definition of a good hatch art can vary widely. However, some common properties of a “good hatch” include [6]:

1. Strokes have a blend of different lengths with a strong preference for longer strokes over shorter strokes
2. Strokes usually are parallel to one another for each layer of hatch strokes
3. Strokes are roughly evenly distributed for the same tone area throughout the entire hatch image because different tones are conveyed by the proximity of the strokes (closely distributed strokes mean more strokes in one area, making the area looks darker)

For these reasons, the quality of a TAM texture can be defined by the general uniformity of the stroke distribution. The proposed Literal Tone and Gaussian Pyramid Differences are indications of uniformity with respect to current texture resolution and tone value; and do not provide an absolute measurement of distribution quality. To generate a high-quality TAM texture, we followed the Praun et al. approach [10] and select a best candidate from a randomly generated set of 1000.

A general rule for setting the random set size is that the lighter the required tone, the larger the set size should be. This is because when the tone is bright, there are fewer strokes, and the evenness of the strokes becomes more important. Praun et al. set the candidate number for the lightest tone as 1000 and gradually reduced it to 100 [10]. Based on empirical trials, random set size of 1000 consistently results in quality TAM textures. With the relatively low computation cost and quick turnaround time of few minutes, the size of 1000 selected. Some statistics are recorded for verifying the computation of the two indexes discussed above. The discussion can be found in Appendix.

4.3 DETAIL OF TAM TONE

The targeted tone of the columns increases towards the right where the increment can be customized. Due to the non-linear human eye response, instead of a constant this increment follows the gamma function presented in Praun et al.'s TAM [10]. Figures 4.2 to 4.5 shows the four generated sets of TAMs for comparison. Figures 4.2 is the set based on linear tone values. The rest are based on the gamma-corrected tone values. The original TAM from Praun et al. has 6 columns. We choose to create 16 columns because today's GPU has more capability and we would like to further generalize the results from Praun et al [10].

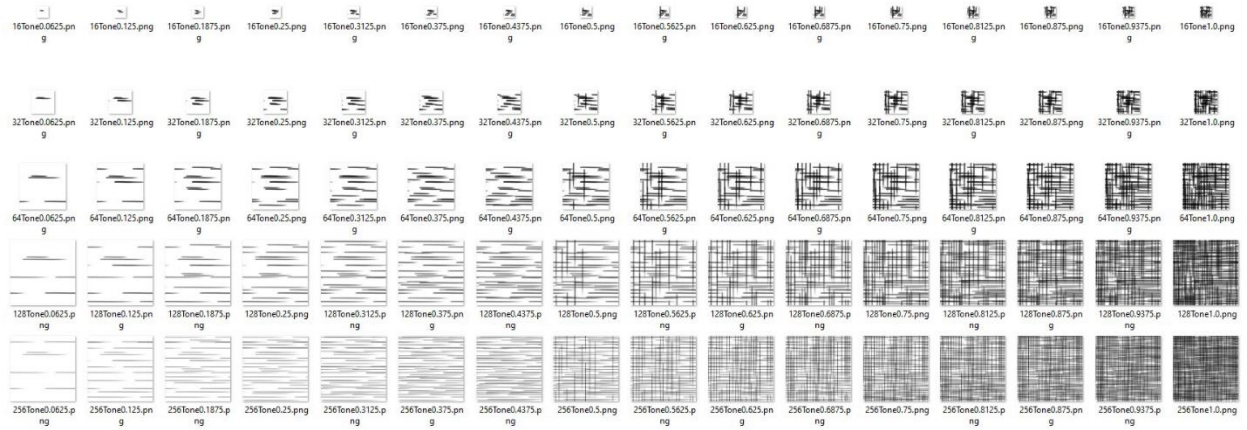


Figure 4.2 16 Linear tone columns 90° cross hatching TAM with 5 mipmap levels

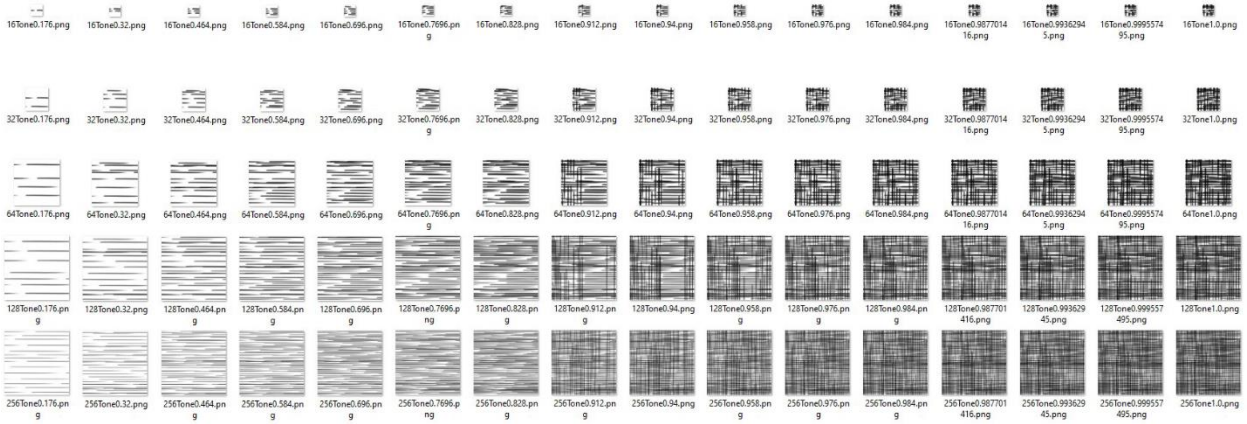


Figure 4.3 16 gamma corrected tone columns 90° cross hatching TAM with 5 mipmap levels

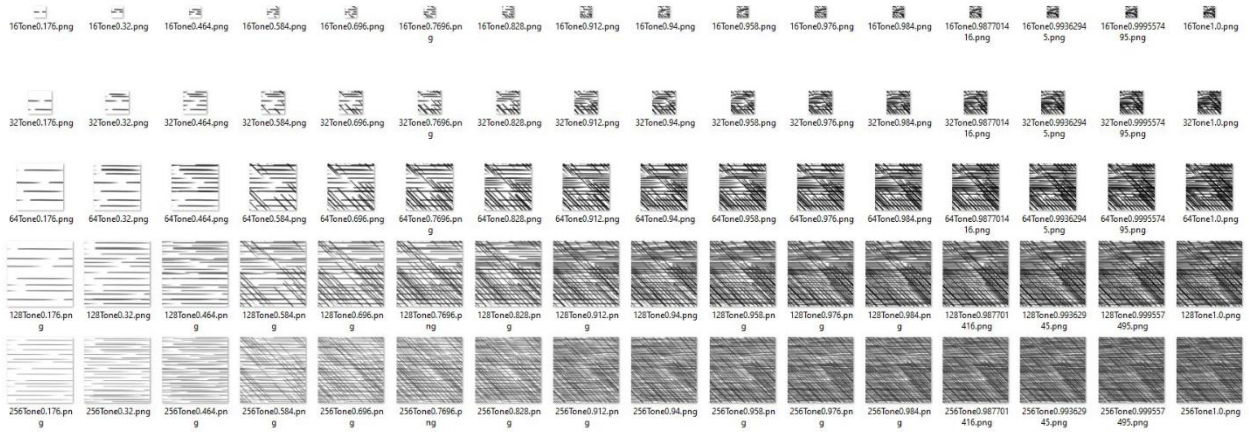


Figure 4.4 16 gamma corrected tone columns 45° cross hatching TAM with 5 mipmap levels

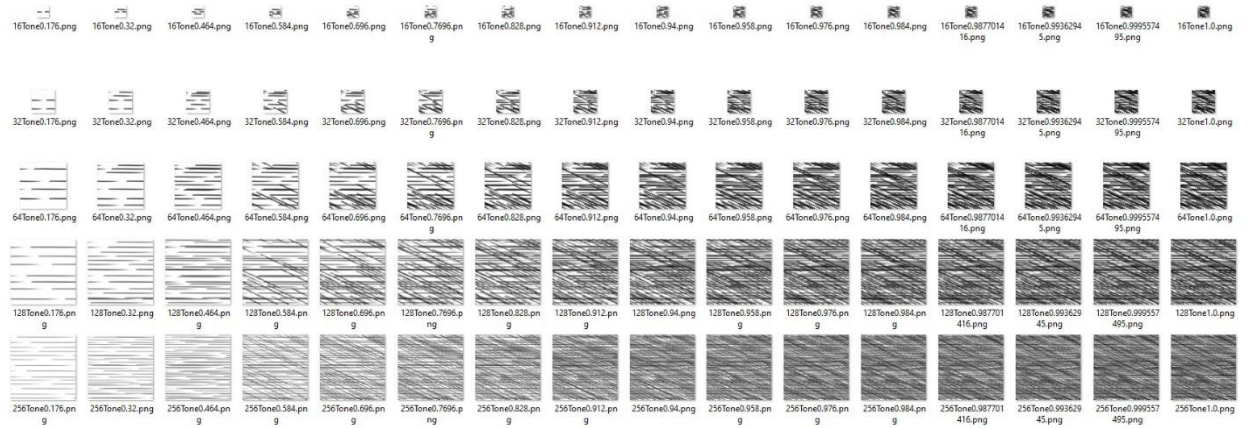


Figure 4.5 16 gamma corrected tone columns 25° cross hatching TAM with 5 mipmap levels

The Tone of a texture is computed as the percentage of nonwhite pixels (nonwhite pixels/all pixels)

Chapter 5. HATCH RENDERING IMPLEMENTATION

The second component of the approach from Praun et al. [10] is **multi-texture blending shading**. The initial version of our implementation was based on that from Lele Feng [11]. This approach properly captures what one would expect to observe in hatch drawing including object silhouette and strokes gradually and smoothly fading in and out as illumination varies across the surface of an object. This is achieved by blending TAM textures from adjacent columns as illumination changes where the texture shown on any position is always a continuous blended result instead of discretized choice.

Praun et al. [10] and Lele Feng [11] performed TAM textures sampling based on percentage of reflected light and are only capable of generating results monochromatically in black and white. Our implementation follows the popular Phong Illumination Model and define the shading of an object to be based on the diffuse and specular components. More importantly, to support artistic creativity, instead of defining a rigid mathematical way of combining the various illumination components, our model allows the user options to compose the illumination result including the capability of integrating object colors. This approach is of particular interest as the actual color of an object can be an important factor affecting hand drawn hatch art.

Praun et al. [10] and Lele Feng [11] both performed the illumination computation in the vertex shader. We generalized the approach and investigated performing this computation in the fragment shader to achieve a higher degree of illumination accuracy. The results and tradeoffs of per-vertex and per-fragment computation will be discussed.

5.1 ILLUMINATION MODEL

Diffuse Component is calculated as follows:

$$diff = max(0, worldLightDirection \cdot worldNormal) * lightStrength$$

The diffuse component is determined by the angle between the worldLightDirection and worldNormal vectors. worldLightDirection is the direction from the light source; and worldNormal vector is the normal at the vertex or the fragment. lightStrength is the attenuated strength of light, which will be discussed in the below section.

Specular Component is calculated as follows:

$$spec = max(0, reflectionDirection \cdot viewDirection)^{specIndex} * lightStrength$$

The specular component is determined by the angle between the relfectionDirection and viewDirection. relfectionDirection is the reflection direction of the light source; and viewDirection is the viewing vector. The specIndex controls the range of the specular highlight, and this parameter is exposed for user manipulation.

The strengths of point light are attenuated according to the distance, d, from the illuminated position from the light,

$$attenuation = smoothstep\left(0.0, 1.0, 1.0 - \left(\frac{n * n}{d * d}\right)\right)$$

n is the distance where attenuation begins.

Lastly, the user has control over the strength and color of the **ambient component**.

5.1.1 *Multiple Light Sources*

The approach from Praun et al.[10] and Lele Feng [11] only support one mono-chromatic illumination source. To further generalize these approaches, we need to support multiple general light sources. The brightness contribution of each light source is computed at every vertex or fragment. A max or addition of all the light's brightness contribution is found as hatch lookup value. After the hatch lookup process, the appropriate hatch texture is selected, and their blending weights are prepared for this particular vertex or fragment. Besides this hatch texture, each light has some other impacts on this vertex or fragment, such as this light's color, intensity, and strength. We compute each light's above other contributions to this vertex or fragment by finding the product among the above properties and the hatch color. Then the final color of this vertex or fragment is the addition of all the lights' contributions.

5.2 HATCH LOOKUP VALUE

Instead of a rigid mathematical model, users have multiple options to combine the computed components to perform the actual TAM texture look up:

1. Users can choose to use diffuse factor directly as hatch lookup value.
   Within this option, users can choose to:
   a. Take the maximum of all the diffuse factors from all light sources
   b. Or take the addition of all diffuse factors from all light sources
   Although taking addition aligns the most with real-world physics, taking the max can be helpful when users want to capture more illumination details from the lights.
2. Users can choose to use specular factor directly as hatch lookup value.
   Within this option, users can choose to:
   a. Take the maximum of all the diffuse factors from all light sources
   b. Or take the addition of all diffuse factors from all light sources
   Although taking addition aligns the most with real-world physics, taking the max can be helpful when users want to capture more illumination details from the lights.
3. Users can choose to use the sum of the diffuse and specular components as hatch lookup value. This option aligns the most with real-world illumination.
   Within this option, users can choose to:
   a. Take the maximum of all the diffuse factors from all light sources
   b. Or take the addition of all diffuse factors from all light sources
   Taking the max can be helpful when users want to capture more illumination details from the lights.

All the above options help users to customize the final hatch rendering result based on their unique needs.

5.2.1 *Color Hatching*

Our shading process is generalized to allow user the options of selecting the foreground (fg) and background (bg) colors for hatching:

$$hatchColor = hatchColor * bg + (1 - hatchColor) * fg$$

5.2.2 *Consideration of Object Color*

Praun et al. [10] and Lele Feng [11] do not support the consideration of object color. However, as discussed above, in hatch arts the color of the object can also be an important factor. To support this feature, we provide an option of integrating object color into the illumination model for TAM texture selection. The color of an object scales the illumination results computed. In this way, darker or more saturated color will result in the selection of darker-tone TAM texture. This aligns with the hand drawn hatch art. The intricacy of artistic interpretations of colors and the effect on hatch texture selection is an interesting future investigation.

Lastly and importantly, unlike hand drawn hatch art which are normally monochromatic. Our flexible illumination model allows user the option to display the hatch results on actual object colors. Though colored hatch art is beyond traditional hatch artwork, we started to see more and more investigation among artists. Figure 5.1 are two good examples of colorization of hand drawn hatch art.

Figure 5.1 a) Colored hatch art from KatarzynaGagolART [21] and b) Angelina Benedetti [22]

5.2.3 *Hatch of Surface Texture*

Our illumination model supports texture mapping on objects, where the user has the option of applying the surface texture as is or the hatched surface texture on the texture.

Chapter 6. RESULTS

In this chapter, we will present the parameters available for user configuration, and all rendering effects. Additionally, we will evaluate these results based on the specification defined in Chapter 3. Results will be presented by first discussing user control parameters, then, the effects of static rendered images, followed by animation, and finally, the rendering effect differences between per-fragment shader method and per-vertex shader method.

For comparison and examination of illumination results, the option of traditional rendering without hatching is also supported (Figure 6.10, 6.11, 6.12 demonstrates this option).

6.1 CONTROL PARAMETERS

One of our project goals is to provide convenient user interface that allows the user to customize the hatch rendering results. Therefore, we exposed many control parameters for user configuration. Figure 6.1 and 6.2 shows the user interface with the exposed parameters in Unity editor.

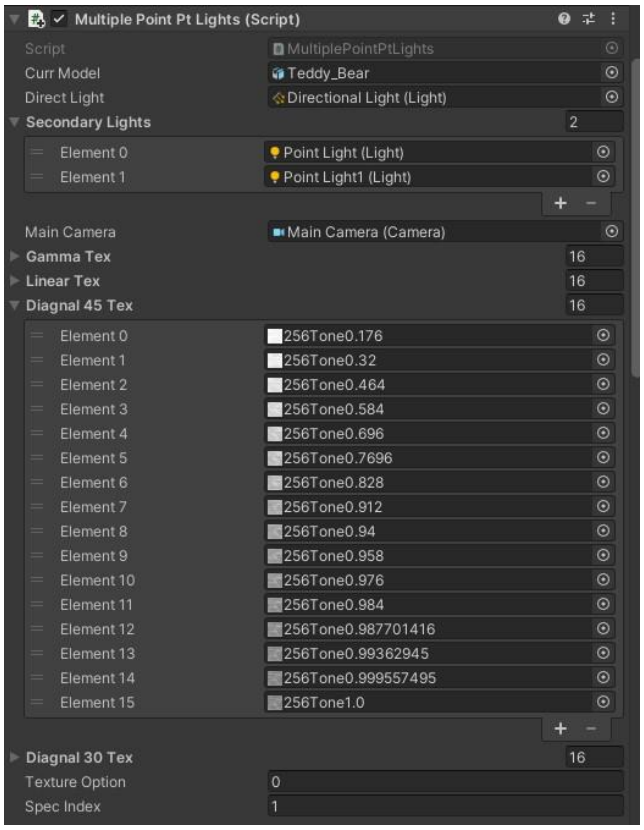


Figure 6.1 CPU side Parameters

Figure 6.1 shows the CPU side parameters available for user manipulation. Users can define their own lights. Currently, the project supports one directional and two point light sources. Users can customize the light properties such as intensity and light color. Current project also accepts 4 sets of TAMs which can be switched at run time. Users can supply custom generated TAM. The Texture Option field is the switch that controls the TAM set to be used. Spec Index is the specular index.

Figure 6.2 GPU side parameters

Figure 6.2 shows the GPU side parameters exposed for user manipulation. Users can customize the tile number of the texture, object color, object outline width, hatch rendering method, hatch lookup value computation options, rendering with or without hatch, underlying texture options, ambient light strength, background and foreground color of colored hatch. In the following, Figure 6.3 shows a user choice option table shows all the possible options users can choose.

New table

	Diffuse Illumination	Specular Illumination	Diffuse + Specular	Ambient Index	Specular Index
Taking Max	HatchRenderMethod = 0 HatchDiffOrSpec = 1 MaxOrAddForLookUpVal = 0	HatchRenderMethod = 0 HatchDiffOrSpec = 0 MaxOrAddForLookUpVal = 0	HatchRenderMethod = 1 HatchDiffOrSpec = any MaxOrAddForLookUpVal = 0	0 - Infinity	0 - Infinity
Taking Addition	HatchRenderMethod = 0 HatchDiffOrSpec = 1 MaxOrAddForLookUpVal = 1	HatchRenderMethod = 0 HatchDiffOrSpec = 0 MaxOrAddForLookUpVal = 1	HatchRenderMethod = 1 HatchDiffOrSpec = any MaxOrAddForLookUpVal = 1	0 - Infinity	0 - Infinity
Object Color Not Considered	HatchRenderMethod = 0 HatchDiffOrSpec = 1 MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0	HatchRenderMethod = 0 HatchDiffOrSpec = 0 MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0	HatchRenderMethod = 1 HatchDiffOrSpec = any MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0	0 - Infinity	0 - Infinity
Object Color Considered	HatchRenderMethod = 0 HatchDiffOrSpec = 1 MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 1	HatchRenderMethod = 0 HatchDiffOrSpec = 0 MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 1	HatchRenderMethod = 1 HatchDiffOrSpec = any MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 1	0 - Infinity	0 - Infinity
Not Show Object Color	HatchRenderMethod = 0 HatchDiffOrSpec = 1 MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0/1 Show/NotShow the Obejct Color = 0	HatchRenderMethod = 0 HatchDiffOrSpec = 0 MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0/1 Show/NotShow the Obejct Color = 0	HatchRenderMethod = 1 HatchDiffOrSpec = any MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0/1 Show/NotShow the Obejct Color = 0	0 - Infinity	0 - Infinity
Show Object Color	HatchRenderMethod = 0 HatchDiffOrSpec = 1 MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0/1 Show/NotShow the Obejct Color = 1	HatchRenderMethod = 0 HatchDiffOrSpec = 0 MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0/1 Show/NotShow the Obejct Color = 1	HatchRenderMethod = 1 HatchDiffOrSpec = any MaxOrAddForLookUpVal = 0/1 ObjectColorInLookUpVal = 0/1 Show/NotShow the Obejct Color = 0	0 - Infinity	0 - Infinity

TAM Variations

Hatch Foreground Color

Hatch Background Color

Underlying Surface Texture

Light's Color

Light's Intensity

Without Hatch

The Render without hatch option has the same option table as this table with hatch rendering. Since this is not the focus of this project, and is a complete replicate of this table, it is omitted.

Figure 6.3 User option table. Options on the right of the table indicates options with no impact to illumination that can be switch on and off for other artistic effects

6.2 TAM RENDERING RESULTS

In this section, we present all rendering results relevant to our TAM structure. We begin with results we reproduced from Praun et al. [10], Lele Feng [11] and Liatis [18]. After which, we present the new rendering results we achieved in this project.

6.2.1 *Smooth Transition During Projected Object Size Changes*

Figure 6.4 shows the camera zoom and the corresponding hatching results. Notice the gradual increase/decrease of hatch strokes, the lengthening/shortening of the existing strokes, while maintaining the same hatch density and hatch stroke size consistency on the head (or torso) of the bear. Our project has properly captured the requirement for transitioning between textures in the same column of a TAM structure to support spatial coherence and maintained hatch art composition homogeneity over spatial changes.

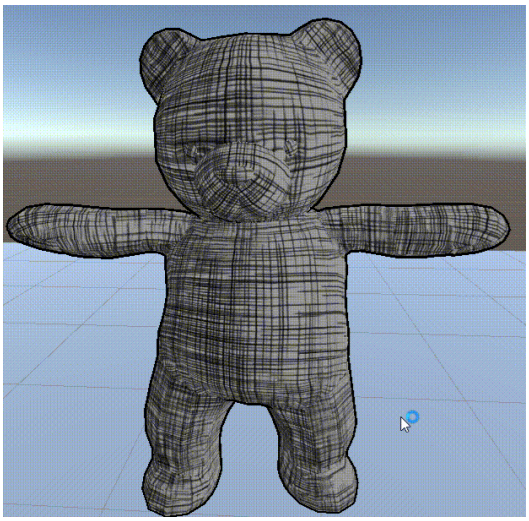


Figure 6.4 Demonstration of smooth transition during projected object size changes

6.2.2 *Smooth Transition During Illumination Changes*

Figure 6.5 shows the light source position change and the corresponding hatching results. Notice the hatch stroke size consistency and the gradual increase/decrease of hatch strokes the head (or torso) of the bear. Our project has properly captured the requirement for transitioning between different columns of a TAM structure and demonstrated and maintained hatch art composition homogeneity over illumination changes.

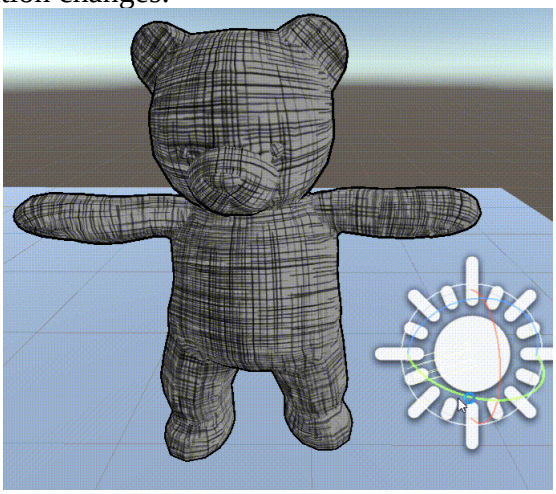


Figure 6.5 Demonstration of smooth transition during illumination changes

6.2.3 *TAM Generalization*

Our generalized TAM process is capable of producing different angled cross hatch which users can switch to at run time. The current project demonstrates 4 different TAMs:

1. 90-degree cross hatching with linear tone setting
2. 90-degree cross hatching with gamma correction
3. 45-degree cross hatching with gamma correction
4. 25-degree cross hatching with gamma correction

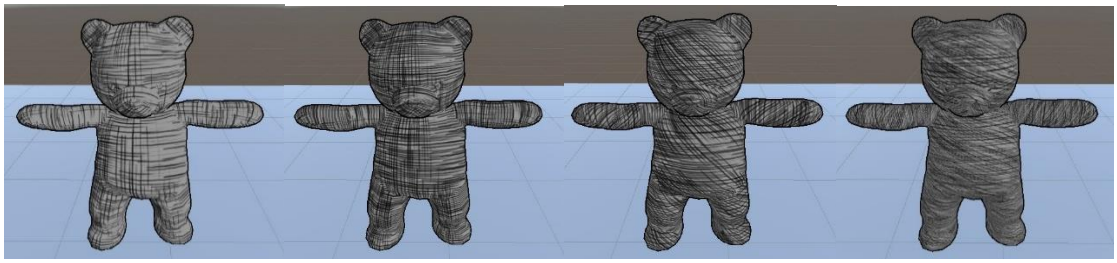


Figure 6.6 The same bear model rendered with 4 different TAM variations. From left to right: 90-degree linear, 90-degree gamma corrected, 45-degree gamma corrected, and 25-degree gamma corrected

6.2.4 *Animations and New Challenges*

Praun et al. demonstrated the rendering effect during illumination change where object rotates or the light source moves. However, no animations with general and movements resulting in object size changes are demonstrated or investigated. These animation conditions are investigated in our project. As Figure 6.8 shows, the illumination variations during the animation results in transitions between TAM columns and are rendered as smooth hatch stroke addition or removal in the rendered images.

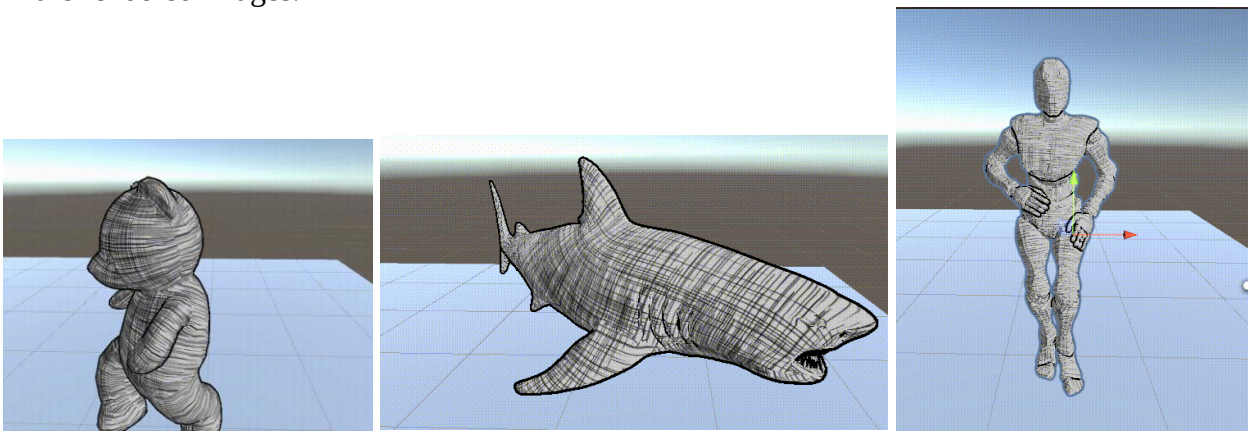


Figure 6.7 Animations on various 3D objects

Though the animations exhibit general smooth transitions during object movements, we discovered a challenge during the examination of the results. Figure 6.9 a shows that when projected object size becomes small, consistency in stroke size may begin to appear too thick and may appear to occupy space that is disproportional to the object size. This leads to the observation presented in Section 2.2 and Section 3.2.4. contradictory spatial coherence requirements:

1. Maintain stroke width and density throughout the scene vs,
2. Allow shorter or thinner strokes on small objects.

These two spatial coherence requirements can also exist in a hand drawn hatch art, where general spatial coherence is required but thinner and shorter strokes are needed for small objects in the scene. The original TAM textures proposed by Praun et al. [10] is designed to address the first requirement but not the second. Interestingly, the second requirement is addressed by the traditional pre-filtered mipmap hierarchy. An approach to address both of these requirements would be integrating TAM and traditional mipmap achieving TAM uniformity of strokes when

projected size are relatively large, and switch to mipmap filtering when projected size of objects becomes small.

6.2.5 *Integration TAM and Traditional Mipmap*

As discussed above, integrating TAM and traditional mipmap unifies the two contradictory spatial coherence requirements. At an appropriate resolution level, we switch TAM texture to traditional mipmap texture, where shorter and thinner strokes are used. The choice of the boundary where the smaller objects can switch to traditional mipmap can be based on artistic opinion. Figure 6.8 shows examples integrated TAM and mipmap where the switching occurred at level 3, 8x8 (Figure 6.8a) and 5, 32x32 (Figure 6.8b). The blurrier results of mipmap filtering are especially obvious in Figure 6.8b.

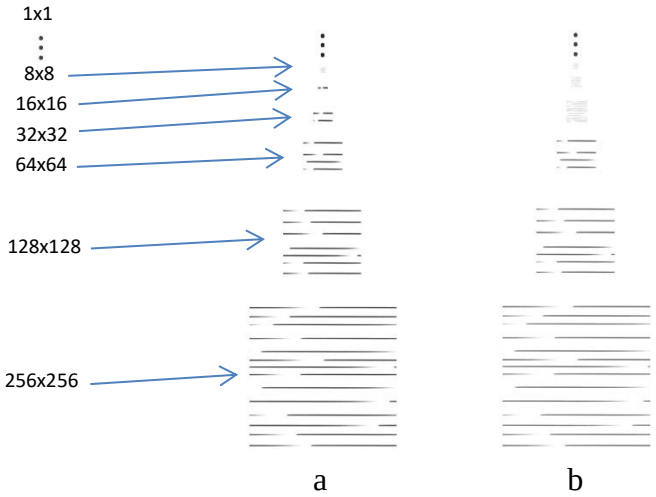


Figure 6.8 TAM switch over to traditional mipmap at a) level-3 8x8; b) level-5, 32x32

As shown in Figure 6.9, 6.9a) uses the 90° cross hatching TAM which switches to traditional mipmap at level-3 (8x8). Figure 6.9b) uses the 90° cross hatching TAM which stops at level-5 (32x32). As expected, 6.9 a) model shows too thick and coarser strokes disproportional to the model size when the projected size is very small. On the other hand, switching from TAM to mipmap at a higher level (Figure 6. 9b) results in softer and blurrier hatches. The visual appears as relatively thinner and shorter strokes proportional to the model size are used on the object when the projected object size is small.

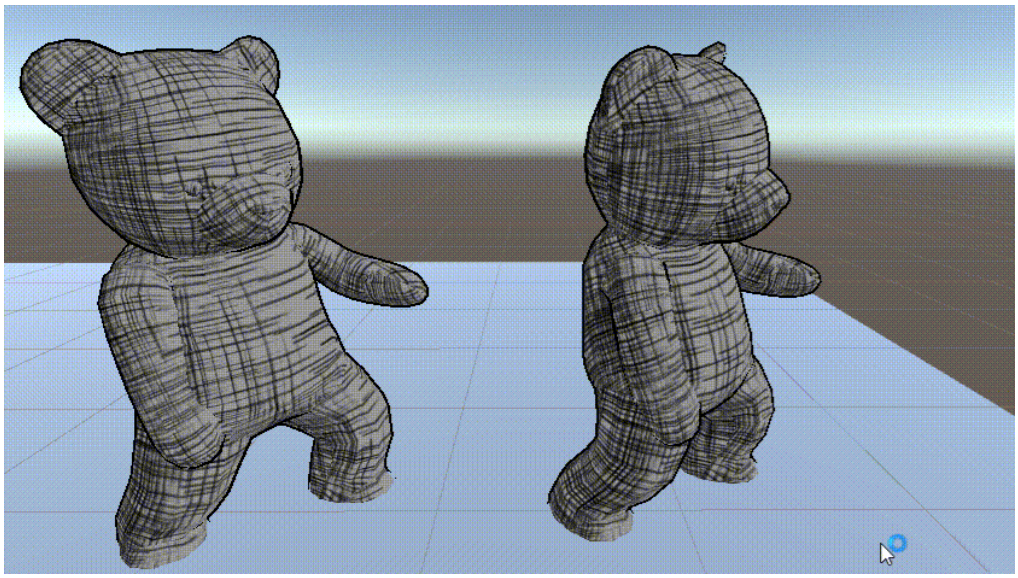


Figure 6.9 Per-vertex shader with a) TAM switch over at level-3; b) TAM switch over at level-5

6.3 ILLUMINATION RESULTS

This project reproduces results from Praun et al. [10] and Lele Feng [11] and further generalizes the results. Following sections presents all the rendering effects resulting from different illumination computation.

This project supports typical illumination effects available for user to choose at run time:

1. Figure 6.10 shows the only **diffuse illumination** with ambient lighting. Under this option, users can further customize the computation method of the diffuse factor during rendering process:
    a. Take the maximum of all lights' diffuse factor as hatch lookup value
    b. Take the addition of all lights' diffuse factor as hatch lookup value

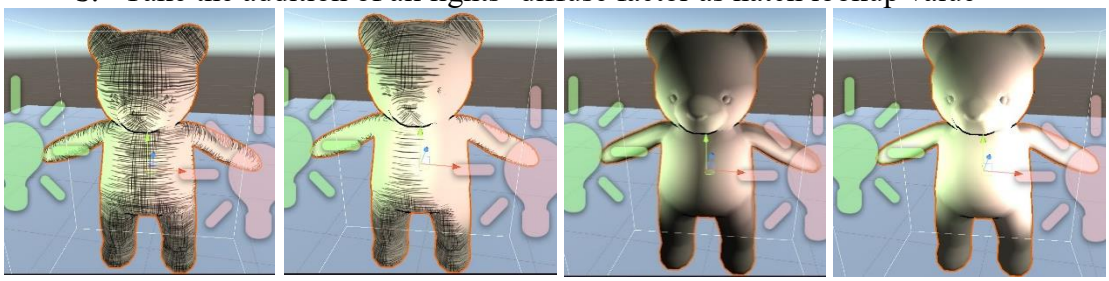


Figure 6.10 The same bear model rendered with only diffuse illumination. From left to right: take maximum of all the light diffuse value as hatch lookup value, take addition result of all the light diffuse value as hatch lookup value. Last two screenshots show the same rendering effects as the first two, only without hatching

Figure 6.10, 6.11, 6.12 all have the same configuration of lights. They all have 1 directional light set to light yellow, two point lights, left one being light green, right one being pink. Lights' position, strength, color maintain unchanged throughout these figures. As we can see, the bear model shows a yellow tint from directional light. The left side of the bear appears green because

of the left point light. The right side of the bear appears pink because of the right point light. All contributions from lights are incorporated appropriately as discussed in implementation chapter.

The second option of taking the addition of all lights' diffuse factor as hatch lookup value makes more sense physically since light in real life is additive. On the other hand, though the first option of taking the maximum does not quite align with physics, it can be helpful artistically. When the user wants to capture more illumination details from every single light source, taking the max from all the light sources helps.

2. Figure 6.11 are results with **specular illumination** and ambient lighting. Under this option, users can further customize the computation method of the specular factor during rendering process:
    a. Take the maximum of all lights' specular factor as hatch lookup value
    b. Take the addition of all lights' specular factor as hatch lookup value

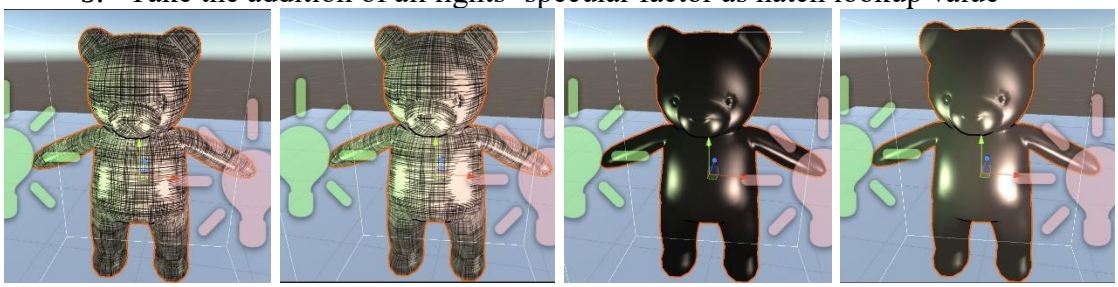


Figure 6.11 The same bear model rendered with only specular illumination. From left to right: take maximum of all the light specular value as hatch lookup value, take addition result of all the light specular value as hatch lookup value. Last two screenshots show the same render rendering effects as the first two, only without hatching

Specular highlight is supported to be moving with the camera position as this is aligned with real world physics.

3. Figure 6.12 shows the results of **diffuse + specular** as the hatch lookup value illumination with ambient lighting. Under this option, users can further customize the computation method of the hatch lookup value during rendering process:
    a. Take the maximum of all lights' diffuse + specular factors as hatch lookup value
    b. Take the addition of all lights' diffuse + specular factor as hatch lookup value

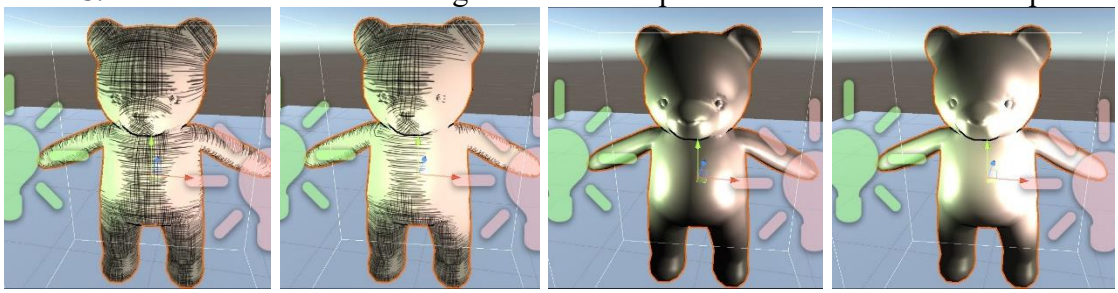


Figure 6.12 The same bear model rendered with diffuse + specular illumination. From left to right: take maximum of all the light diff+spec value as hatch lookup value, take addition result of all the light diff+spec value as hatch lookup value. Last two screenshots show the same

6.3.1 *Colored and Multiple Light Sources*

To further generalize the Praun et al.'s [10] original approach, this project supports colored light sources. Figure 6.13 shows the support of colored point and directional lights. It is interesting to

note that colored hatched art is a novelty and is gradually being investigated by artists and it is not commonly seen traditionally.

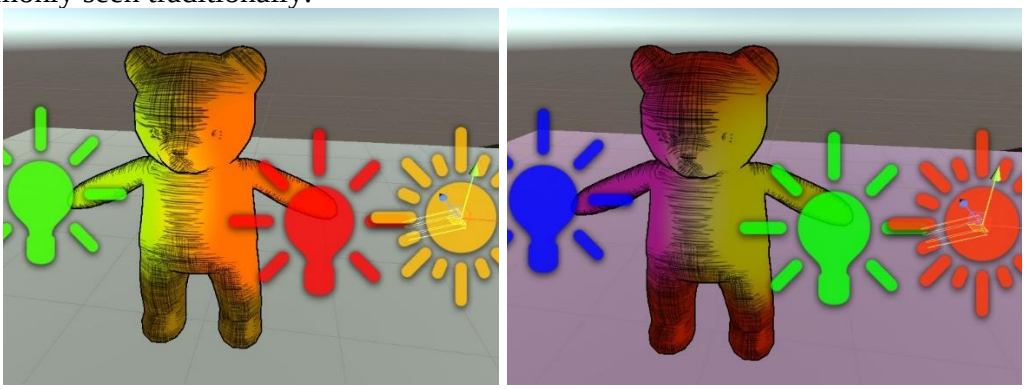


Figure 6.13 Bear model rendered under colored directional and point lights, keeping all other setting the same

6.3.2 *Colored Hatch*

Though relatively novel, colored hatch arts are becoming available. Therefore, we extended previous results and supported colored hatch. Hatch background and foreground color are both parameters for user customization. Figure 6.14 shows two examples setting different background and foreground color, with all other setting unchanged.

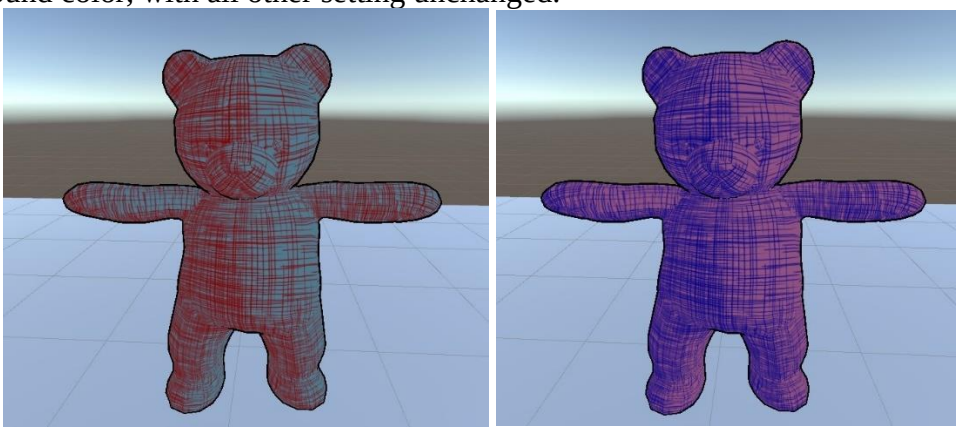


Figure 6.14 Colored hatch examples

Additionally, as above results demonstrates, multiple light sources is supported to further generalize the approach adopted from Praun et al [10].

6.3.3 *Object Color*

As discussed Praun et al. [10] does not consider color, where our illumination model provides the option to integrate object color in hatch lookup as the information can be an important variable in hand drawn hatch art. Figure 6.15 shows examples of three different model with different object color when monochromatically hatched. Figure 6.15 a does not consider object color into hatch lookup value. Figure 6.15 b considers object color into hatch lookup value with a white object color setting. Figure 6.15 c considers object color into hatch lookup value with a brown object color setting. Notice Figure 6.15 a and b shows the same rendering effects and c

shows darker hatch texture on the model. The darker hatch on the c model conveys the sense of a darker colored bear comparing to b model even when the scene is black and white hatch.

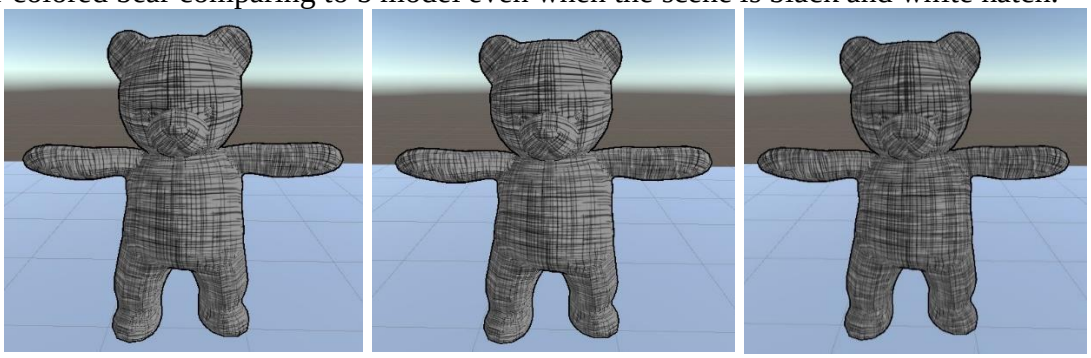


Figure 6.15 Three bear models all have no object color shown, with a) Object color not considered into hatch lookup value; b) Object color White considered into hatch lookup value; c) Object color Brown considered into hatch lookup value

Besides considering object color into the hatch texture selection process, actual object color can also be an important artistic component in hatch nowadays. Figure 5.2 a is a good example of using object color and light color to compose a great hatch art. Therefore, we also expose the options to show object color and light source color. Figure 6.16 shows examples of hatch results considering the object color, and displaying, a brown color on the final rendering along with all the colors from the light sources.

Figure 6.16 Same object color considered into hatch lookup value and the object color is shown

6.4 UNDERLYING TEXTURE

As discussed, our shader supports typical texture mapping with two options:

1. Display as original texture
2. Displayed as TAM hatch

The option of including texture map is available to the user to be applied onto the 3D model. The user has the additional option of either displaying this texture directly or enabling TAM hatch on the texture.

Figure 6.17 Model rendered with original texture applied directly

Figure 6.18 Model rendered with texture as hatch

6.5 FURTHER INVESTIGATIONS

6.5.1 *Per-Fragment vs Per-Vertex Illumination*

Modern GPU offers significant computation capability which allows the illumination computation to be carried out on a per-pixel (fragment) bases. Figure 6.19 shows that the tone transition of per-fragment computation results in clearer hatch strokes and smoother fading in and out of the strokes. This is because the more accurate illumination computation occurs at every pixel. In contrast, per-vertex shader computes the illumination at every vertex position and interpolates the results for per-pixel TAM hatch lookup. The interpolation of illumination ensures a gradual change in value which results in blurring and softening of the results from TAM texture sampling.

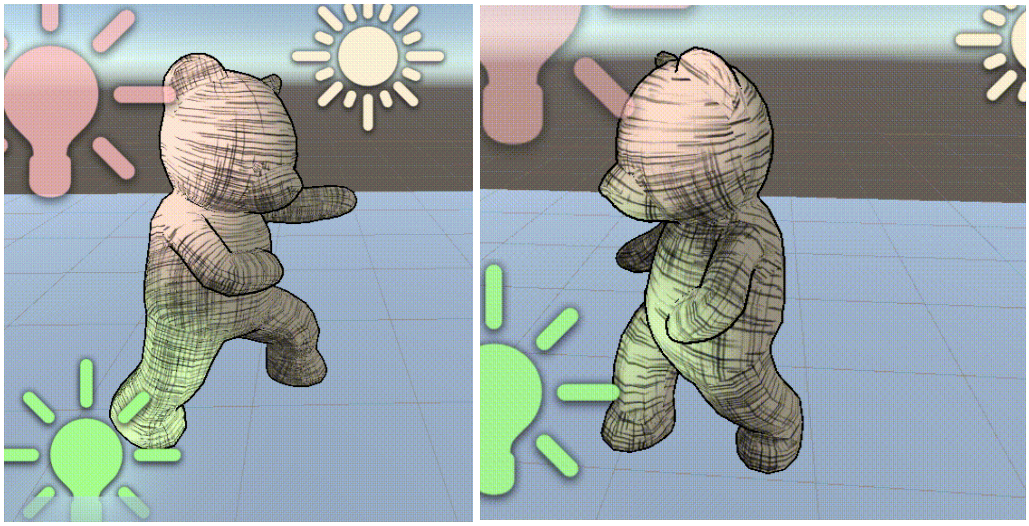


Figure 6.19 a) Per-fragment shader method vs. b) Per-vertex shader method

Though per-fragment shader method has the advantage of sharper hatch strokes and smoother tone transition, as Figure 6.20 shows, the accurate per-pixel computation is capable of capturing rapid illumination variations. This rate of variation may exceed the neighboring column tone differences of the TAM structure resulting in TAM column skipping. This is especially the case when illumination varies rapidly around small projected area. In the case of Figure 6.20 a), the TAM column skipping resulted in discontinuity of strokes showing as dots around rapidly changing illumination boundaries. In this case, neighboring pixels can have very different illumination resulted in distinct TAM texture column being sampled for a given region. This results in texture aliasing appearing as dots on the object.

In contrast, as Figure 6.20b shows, per-vertex method does not exhibit such aliasing effect. During per-vertex illumination, the interpolated illumination values ensure smoother transitions between neighboring pixels resulting in gradual transitions between TAM columns being sampled. The gradual tone changes and smoother hatch results suggest per-vertex shaders are more suitable to support animations when the projected object size is likely to vary over time.

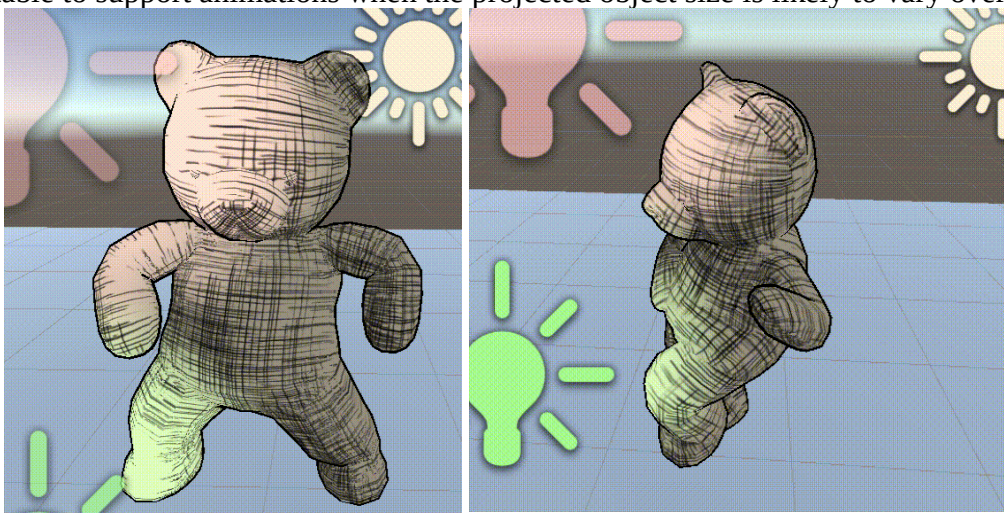


Figure 6.20 a) Per-fragment shader method with TAM; b) Per-vertex shader method with TAM

In summary, our project supports user configurable parameters for customizing the final rendering results. We successfully reproduced the results from Praun et al [10]. The TAM

generation is fully automated and supports customization of hatching. The successful TAM construction ensures the smooth transition during object size and illumination changes. We further support a more general illumination models including specularity, light source types, variable number of light sources, variable hatch, light and object colors, and rendering of surface textures as cross hatch. Additionally, we have investigated and proposed configurations for proper animation support. We further investigated per-fragment and per-vertex illumination computation differences and identified the suitable situations respectively. The novel integration of TAM and traditional mipmap ensure the fulfilling of potentially contradictory spatial requirements.

Chapter 7. CONCLUSION AND FUTURE DIRECTION

7.1 Learning Outcomes and Contributions

In this project, we studied the current approaches of hatch rendering, selected the most appropriate method of real-time hatching presented by Praun et al. [10]; And we designed, implemented, and further generalized this method to reproduce an interactive real-time hatch rendering system. Our implementation begun with the results from of Liatis [18] and Lele Feng [11].

As a crucial component of Praun et al. [10]'s method, we followed the TAM generation process, which supports various hatch styles. We ensured appropriate TAM structure so that it supports smooth transition during zoom changes. We also ensured uniform distribution of strokes required by hatch art. We have discovered the issue of potentially contradictory spatial coherence requirements during animations and hand-drawn hatch art; and we demonstrated a viable solution based on integrating TAM and traditional mipmap.

Another important component of Praun et al. [10]'s approach is multi-texture blending method. The multi-texture blending method achieved to support appropriate hatch texture selection based on illumination and smooth transition during illumination changes.

We also generalized the solution to support a more complete illumination model, including specularity, light source types, variable number of light sources, variable hatch, light and object colors, and rendering of surface textures as cross hatch. While the original solution only performs illumination computation and hatch lookup in vertex shader, we investigated illumination computation and hatch lookup in fragment shader. We examined the differences of rendering result and tradeoffs between the two methods. While per-fragment shader and is capable of capturing accurate illumination variations. It is also the case that when illumination varies rapidly, this approach is prune to texture aliasing artifacts. Based on these results, one should always adopt per-vertex solution in case of animations to better handle object size changes. In static images, when object sizes are relatively large and constant, one can consider per-fragment solution to better capture the rapid illumination changes.

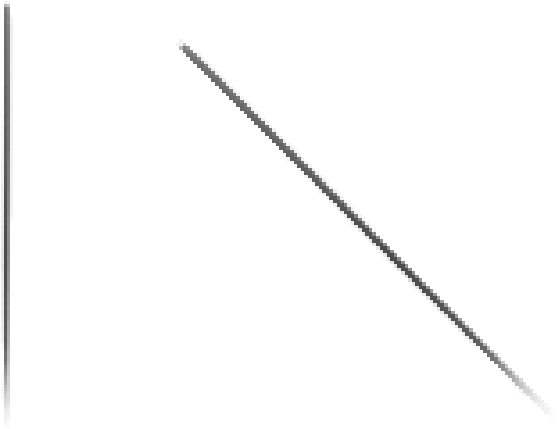
We evaluate the success of the project by flexibility, generalizability, performance, quality of rendering, animation quality and the novel contribution. The project allows options for users to customize the rendering results in real time. The hatch rendering is in real time, and the rendering effect is aesthetically pleasing and resembles hand drawn hatch art. The hatch rendering not only reproduced but also generalized and improved upon results from Praun et al. [10].

In summary, the project generalized the results to support a more sophisticated illumination models including specularity, light source types, variable number of light sources, variable hatch, light and object colors, and rendering of surface textures as cross hatch. The animation results show smooth transition during illumination and object size changes. The novel integration of TAM and traditional mipmap meets the requirements of potentially contradictory spatial coherence. The investigation of illumination computation and hatch lookup in per-vertex vs per-fragment shader revealed the tradeoffs of between the two and shows potential future work opportunities. Based on the specifications, this project successfully achieved the goals.

7.2 LIMITATIONS AND FUTURE WORK

Though the project is successful based on our specifications, there are some limitations to the current project. First, as discussed in results chapter, per-fragment shader and per-vertex shader both has their own advantages and disadvantages. If any approaches that can remedy the disadvantages of the two methods, the final rendering effect will be further improved. Additionally, as discussed in results chapter, the option of taking object color into consideration of illumination factor is provided for user selection. However, the accurate interpretation of a color its corresponding hatch texture was not investigated in detail. Future work can work on top of current options and further improve the interpretation of a color to its corresponding hatch texture.

Future work can also work on antialiasing of the strokes when rotated other than multiples of 90°. When strokes are rotated 90°, no aliasing was observed. Whereas when strokes are rotated to other angles, aliasing is observed as Figure 7.1 shows. This is a limitation of the current method of using pre-drawn raster stroke image. When using raster images, aliasing happens inevitably. Due to the time constraint of this project, we did not investigate anti-aliasing processing on the raster stroke image. If anti-aliasing processing can be applied, the quality of the TAM will be improved.



a b

Figure 7.1 a) stroke rotated 270° for 90° cross hatching texture; b) stroke rotated 315° for 45° cross hatching texture

Current TAM generation is a separate process from the rendering process, consisting of several phases; And the TAM generation code is written in python. This approach has the advantage of quicker development feedback loop and easier modification in either of the phases. On the other hand, it also prevented the entire process from TAM to rendering to be real time. Future work can concentrate on combining the different phases and making the entire process real time. This can make the application more automated and easier to use.

Besides the limitations of the current project, some other interesting works can be integrated to improve the rendering results. Lee et al. adopted similar approaches as Praun et al. [10] to achieve real-time pencil rendering method [23]. Interestingly, as illustrated in Figure 7.2, they were able to make the outline drawing look very natural by drawing the outline multiple times with slightly distorted trajectories. This mimics the hand-drawn trial and error process which is useful in pencil hatching.

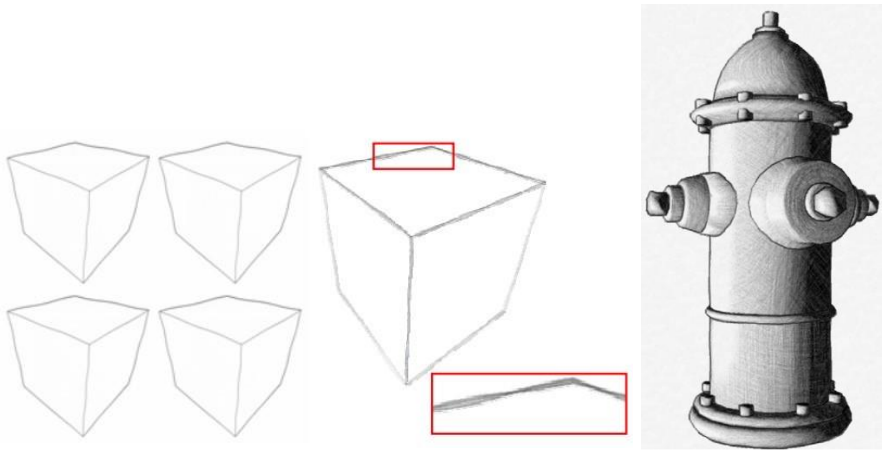


Figure 7.2 One of the final results from Lee et al. [23]

Lastly, there are many other effects that can be generalized for hatch rendering. For example, all the mentioned research above does not include object shadow. Considering hatch art represent shadows also as hatch, this could be an interesting future work. Other examples include, light glow, fog, etc. While the implementation might seem straightforward, similar to our experience from animation and per-fragment illumination investigations, interesting and unforeseen issues can always appear.

BIBLIOGRAPHY

[1] B. Gooch and A. Gooch, *Non-Photorealistic Rendering*, 0 ed. A K Peters/CRC Press, 2001. doi: 10.1201/9781439864173.
[2] "Photorealistic Rendering. You Have Options," *ArchiCGI*, Apr. 05, 2016. https://archicgi.com/architecture/photorealistic-rendering-options/ (accessed Jan. 16, 2022).
[3] "zdesignviz: npr," *zdesignviz*. https://zdesignviz.blogspot.com/p/npr.html (accessed Jan. 16, 2022).
[4] "NPAR Conference - Home." https://dl.acm.org/conference/npar (accessed Jan. 01, 2022).
[5] "Guide to Shading Techniques: Hatching, Cross-Hatching, Scribbling and Others," *Erika Lancaster-Artist, Content Creator & Online Art Teacher*. http://www.erikalancaster.com/2/post/2017/09/guide-to-shading-techniques-hatching-cross-hatching-scribbling-and-others.html (accessed Jan. 09, 2022).
[6] "Hatching and Cross Hatching Drawing Techniques." https://thevirtualinstructor.com/hatchingcrosshatching.html (accessed Jan. 09, 2022).
[7] "6 Basic Forms of Hatching and Cross Hatching," *Craftsy*. https://www.craftsy.com/post/hatching-and-cross-hatching/ (accessed Jan. 17, 2022).
[8] A. L. Guptill, *Rendering in pen and ink*, Paperback ed., 1 print 1997, [Nachdr.]. New York: Watson-Guptill, 2009.
[9] "Art of Kristen Maduik: Image," *Tumblr is a place to express yourself, discover yourself, and bond over the stuff you love. It's where your interests connect you with your people.* https://78.media.tumblr.com/0ff7a8ee2e1703661faaf493f331ca7b/tumblr_pfdpp8qPTl1uplirco1_1280.jpg (accessed Jan. 17, 2022).
[10] E. Praun, H. Hoppe, M. Webb, and A. Finkelstein, "Real-time hatching," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques - SIGGRAPH '01*, Not Known, 2001, p. 581. doi: 10.1145/383259.383328.
[11] F. Lele, "《Unity Shader 入门精要》源代码." Oct. 07, 2022. Accessed: Oct. 07, 2022. [Online]. Available: https://github.com/candycat1992/Unity_Shaders_Book
[12] P. Bénard, A. Bousseau, and J. Thollot, "State-of-the-Art Report on Temporal Coherence for Stylized Animations," *Comput. Graph. Forum*, vol. 30, no. 8, pp. 2367–2386, Dec. 2011, doi: 10.1111/j.1467-8659.2011.02075.x.
[13] T. Umenhoffer, L. Szirmay-Kalos, L. Szécsi, Z. Lengyel, and G. Marinov, "An image-based method for animated stroke rendering," *Vis. Comput.*, vol. 34, no. 6–8, pp. 817–827, Jun. 2018, doi: 10.1007/s00371-018-1531-9.
[14] "Pyramidal parametrics | ACM SIGGRAPH Computer Graphics." https://dl.acm.org/doi/10.1145/964967.801126 (accessed Dec. 10, 2022).
[15] Steve Marschner *et al.*, *Fundamentals of Computer Graphics*. Accessed: Dec. 09, 2022. [Online]. Available: https://learning.oreilly.com/library/view/fundamentals-of-computer/9781482229417/K22616_C011.xhtml
[16] "(37) Pinterest," *Pinterest*. https://www.pinterest.com/pin/18084835990115621/ (accessed Oct. 16, 2022).
[17] "Hendrick Goltzius | Marcus Valerius, from the series The Roman Heroes," *The Metropolitan Museum of Art*. https://www.metmuseum.org/art/collection/search/343575 (accessed Nov. 20, 2022).

[18] H. Liatis, “CS7490 Final: Real-time Hatching.” https://sites.google.com/site/cs7490finalrealtimehatching/home (accessed Oct. 23, 2022).
[19] Zeno, “Werk: »Beham, Hans Sebald: Junger Herr und Mädchen« aus der Sammlung »40.000 ...” http://www.zeno.org/Kunstwerke/B/Beham,+Hans+Sebald%3A+Junger+Herr+und+M%C3%A4dchen (accessed Jun. 19, 2022).
[20] “Heinrich Aldegrever | Dancing Couple, with the Male Figure Mid-Step, from The Small Wedding Dancers,” *The Metropolitan Museum of Art*. https://www.metmuseum.org/art/collection/search/428734 (accessed Jun. 19, 2022).
[21] “No.1 Colorful Woman Print Unique Wall Art Print Living - Etsy.” https://www.etsy.com/listing/899127275/no1-colorful-woman-print-unique-wall-art?utm_source=OpenGraph&utm_medium=PageTools&utm_campaign=Share (accessed Nov. 19, 2022).
[22] “Keira Knightley Ballpoint Pen by AngelinaBenedetti on DeviantArt.” https://www.deviantart.com/angelinabenedetti/art/Keira-Knightley-Ballpoint-Pen-292975400 (accessed Nov. 19, 2022).
[23] H. Lee, S. Kwon, and S. Lee, “Real-time pencil rendering,” in *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering - NPAR ’06*, Annecy, France, 2006, p. 37. doi: 10.1145/1124728.1124735.

APPENDIX A

Best Stroke Selection Discussion of Statistics

Some statistics are recorded for verifying the computation of the two indexes discussed in Chapter 4.

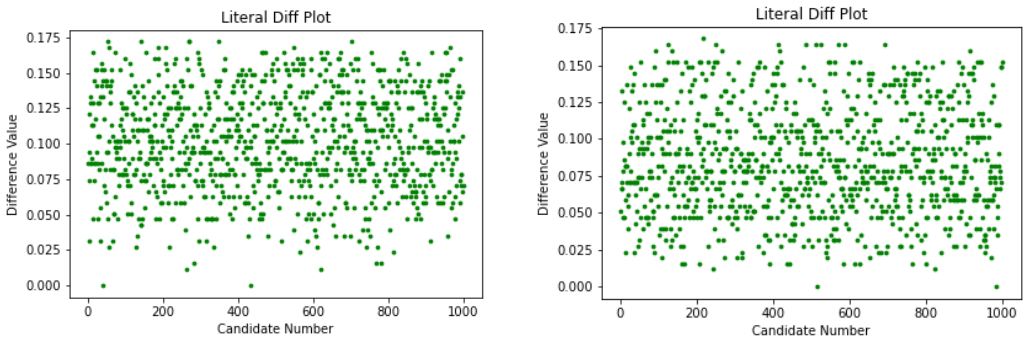


Figure 7.3 Literal tone progress of 1000 candidates of the third stroke and fourth stroke of the first texture of the first tone column

As shown in Figure 7.3, the literal tone progress of 1000 candidates of the third stroke and fourth stroke of the first texture of the first tone column are plotted. The values of the tone progress become smaller for the fourth stroke's candidates compared to the third stroke's candidates. This is expected results since more strokes drawn in the texture would increase the possibility of the new strokes getting overlapped with existing strokes. The overlapping would decrease the tone contribution of the new stroke bring to the texture, which makes the tone progress value smaller. This is observed in every tone column as expected.

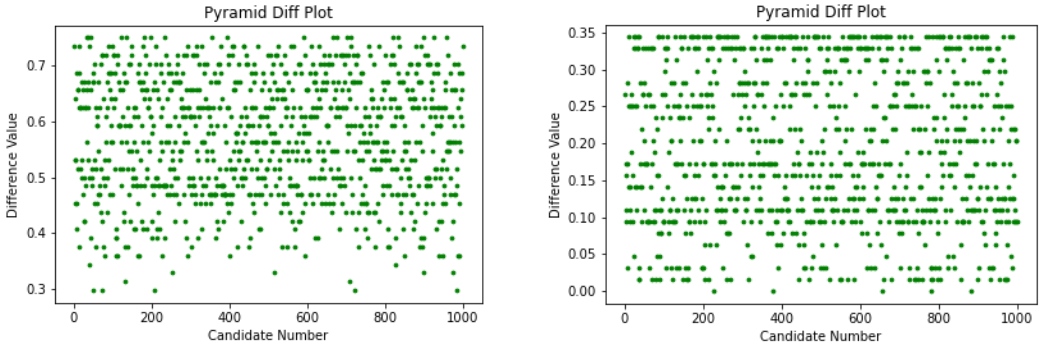


Figure 7.4 Distribution value sum of 1000 candidates of the first stroke and second stroke of the first texture of the first tone column

Figure 7.4 shows the distribution value sum of the 1000 candidates of the first stroke and second stroke of the first texture of the first tone column. The distribution value is gradually decreasing, which is expected behavior of the distribution value. The more strokes existing in the current canvas, the less difference the new stroke is expected to make after Gaussian blurs. This is because Gaussian blur averages the surrounding pixel value into one new pixel, and we view non-white pixels as colored pixels while finding the tone of the texture. More strokes drawn in the texture means more closely strokes reside, which makes tone contribution after the Gaussian processing smaller.

APPENDIX B

Multi-Texture Blending Detail (Hatch Texture Lookup Process)

We take 2 adjacent textures at a time for a vertex or fragment based on its illumination factor. We first break the [0 –ToneLevel+1] range into ToneLevel + 1 intervals. Take this project's setting as an example, we break the tone range into 17 intervals. When the hatch lookup value is in the largest value interval (>16), we do not apply any weight to any of the textures. This means we do not use any of the textures, which makes the final color of this vertex or fragment completely white. We intentionally choose to use white color for the brightest spots on the object, because in hand-drawn hatch art, artists usually use completely white to represent the brightest spots. When the hatch lookup value is in the next largest value interval (16> hatchFactor >15), we blend white and the lightest tone texture, with white taking the hatch lookup value as weight, the lightest tone texture taking (1- white weight) as its weight. Similarly for hatch lookup value falls in the third largest value interval (15> hatchFactor >14), we blend the lightest tone texture and the second lightest tone texture, with the lighter tone texture taking hatch lookup value as weight, the darker tone texture taking (1- white weight) as its weight. This process continues until we covered the entire scope of the hatch lookup value.

As the process demonstrates, two textures (or one being white color) are blended to shade one vertex or fragment at a time. When the illumination condition changes on this spot, hatch lookup value gradually changes; When hatch lookup value gradually changes, the weights of the two textures gradually change accordingly. This ensures the transition of the shading is smooth when hatch lookup value is still in the same tone interval. Furthermore, when hatch lookup value moves to another adjacent tone interval, The transition is still smooth because the two adjacent tone intervals must have a same hatch texture. For example, when illumination on a spot becomes darker, the darker-tone texture of the lighter-tone interval continues to be the lighter-done texture of the darker-tone interval. The visual looks like this common texture gradually fade in with larger and larger weights when the illumination becomes darker. Then this common texture gradually fades out with the third texture gradually fading in representing an even darker tone.