REAL-TIME STYLIZED RENDERING FOR LARGE-SCALE 3D SCENES

A Thesis

presented to

the Faculty of California Polytechnic State University,

San Luis Obispo

In Partial Fulfillment

of the Requirements for the Degree

Master of Science in Computer Science

by

Jack Pietrok

June 2021

COMMITTEE MEMBERSHIP

TITLE: Real-time Stylized Rendering for Large-scale 3D Scenes

AUTHOR: Jack Pietrok

DATE SUBMITTED: June 2021

COMMITTEE CHAIR: Zoe Wood, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Christian Eckhardt, Ph.D.
Professor of Computer Science

COMMITTEE MEMBER: Franz Kurfess, Ph.D.
Professor of Computer Science

ABSTRACT

Real-time Stylized Rendering for Large-scale 3D Scenes

Jack Pietrok

While modern digital entertainment has seen a major shift toward photorealism in animation, there is still significant demand for stylized rendering tools. Stylized, or non-photorealistic rendering (NPR), applications generally sacrifice physical accuracy for artistic or functional visual output. Oftentimes, NPR applications focus on extracting specific features from a 3D environment and highlighting them in a unique manner. One application of interest involves recreating 2D hand-drawn art styles in a 3D-modeled environment. This task poses challenges in the form of spatial coherence, feature extraction, and stroke line rendering. Previous research on this topic has also struggled to overcome specific performance bottlenecks, which have limited use of this technology in real-time applications. Specifically, many stylized rendering techniques have difficulty operating on large-scale scenes, such as open-world terrain environments. In this paper, we describe various novel rendering techniques for mimicking hand-drawn art styles in a large-scale 3D environment, including modifications to existing methods for stroke rendering and hatch-line texturing. Our system focuses on providing various complex styles while maintaining real-time performance, to maximize user-interactability. Our results demonstrate improved performance over existing real-time methods, and offer a few unique style options for users, though the system still suffers from some visual inconsistencies.

ACKNOWLEDGMENTS

Thanks to:

- My family, Ted, Kim, and Jason, for their constant support and encouragement, and for helping me realize my goals
- My advisor, Dr. Zoe Wood, for reminding me to be creative, and for showing me the true power of computer science
- The members of my graphics group, for their endless enthusiasm and insight
- The other members of my thesis committee, Dr. Kurfess and Dr. Eckhardt, for being excellent professors to learn from
- The graphics and computer science communities at Cal Poly, for creating a welcoming and friendly environment for research and learning
- My former mentor Brian Schrom, for introducing me to the possibilities of computer science and engineering
- Andrew Guenther, for uploading this template

TABLE OF CONTENTS

Page

LIST OF TABLES viii

LIST OF FIGURES ix

CHAPTER

1 Introduction 1

1.1 Photorealism vs Non-photorealism 1

1.2 Modern Stylized Rendering 2

1.3 Contributions 3

2 Background 4

2.1 World-Space Versus Screen-Space 4

2.2 Level-of-Detail Techniques 5

2.3 Compute Shaders 7

2.4 Terrain Features 7

3 Related Works 10

3.1 Suggestive Contours 10

3.2 Stroke Rendering 11

3.3 Painterly Rendering 12

3.4 Stylized Texturing 14

3.5 Color & Shading 15

4 Implementation 16

4.1 Curvature-based Strokes 17

4.1.1 Parameterizing Orientation 17

4.1.2 Stylization Options 18

4.2 Hybrid Overlay Strokes . . . 22
4.2.1 ID Reference Image . . . 22
4.2.2 Feature Extraction . . . 23
4.2.3 Chaining Algorithm . . . 25
4.2.4 Stroke Mesh Construction . . . 27
4.2.5 Stroke Rendering . . . 30
4.3 Temporal Coherence . . . 30
4.3.1 Motion Field . . . 30
4.3.2 Adjustments to Smooth Motion Field . . . 31
4.4 Hatching . . . 33
4.4.1 Dynamic Blending . . . 33
4.4.2 Dynamic Solid Textures . . . 34
4.5 Trees . . . 35
5 Validation & Results . . . 38
5.1 Performance Results . . . 38
5.2 Visual Results . . . 41
5.3 User Study . . . 42
6 Future Work . . . 53
6.1 Hatching Limitations . . . 53
6.2 Coherence Limitations . . . 54
7 Conclusion . . . 56
BIBLIOGRAPHY . . . 57
APPENDICES
A Visual Results Survey . . . 62

LIST OF TABLES

Table Page

5.1 Performance results for various stylized rendering methods, across different scene sizes and camera positions. Scenes represented in this table all utilized curvature-based stroke rendering. 40

5.2 Performance results for various stylized rendering methods, across different scene sizes and camera positions. Scenes represented in this table all utilized hybrid-overlay stroke rendering. 40

LIST OF FIGURES

Figure Page

1.1 A side-by-side example of a photorealistic image (in this case an actual photograph) and stylized rendition of the same image. Stylized images generally forgo physical accuracy for improved visual appeal or style. Images obtained under creative commons from http://bjornfree.com/galleries.html 2

2.1 An example of a billboard used to reduce model complexity in large-scale scenes. The image in *b* replaces instances of trees that are far away from the camera. The reduced quality is not as noticeable at long range. 6

2.2 An early iteration of our terrain model, with contours rendered as black lines. 9

3.1 An example mesh with highlighted edge features. Note the improvement in detail when adding contours, and suggestive contours. [9] . 11

3.2 A model of Mt. Rainer, rendered using a painterly style. [4] 13

3.3 An example of real-time hatch textures, generated by overlaying lighter tones on top of themselves at varying intervals. [29] 14

3.4 An example of toon shading, applied with various toon textures, defined in [2]. 15

4.1 Curvature-based strokes utilize the screen-space fragment normal (Ns) and its perpendicular counterpart (Ds) to approximate stroke directionality. 18

4.2 Once the directionality of a stroke has been determined, the space can be roughly parameterized along the length and width of a perceived stroke. In this example, U represents the horizontal stroke dimension, and V represents the vertical stroke dimension. 19

4.3 Two examples of stylized curvature-based strokes. The textures beneath each image are the same used in the images above them. While some warping is visible, the textures mostly face the direction of the perceived strokes. 20

4.4 Contour strokes are rendered in this scene as solid black regions, with slight alpha tapering along the edges. Curvature-based strokes are spatially and temporally coherent, but are limited in what options they provide for stylization. 21

4.5 A sample from a 480x270 ID reference image, used for edge extraction. Each edge is rendered as a unique RGBA value representing its ID. Colors have been exaggerated for demonstration purposes. . 24

4.6 A diagram depicting chaining constraints. A denotes a user-specified angle constraint for chaining, and D denotes a user-specified distance constraint in pixels. 26

4.7 An example of how triangle strips can be formed from the list of vertices for a given chain (right). The other images demonstrate some example stylizations that can be applied to the strips using a shader. [27] 28

4.8 Three vertices are added to the ends of each stroke to create a rounded edge. The red labeled points indicate added vertices. . . . 29

4.9 A depiction of a motion-field used to improve temporal coherence. Each white line representation the direction that the terrain has moved in the last ten frames 32

4.10 An example of hatching with basic mipmapping. Note that surfaces further from the camera become noisy and lose distinct stroke-line visuals 34

4.11 An example of a dynamic solid texture with a checkerboard pattern. Each consecutive image shows a higher level of texture scaling that can be modulated using specialized blending weights. The rightmost image shows the combined dynamic solid texture, to be filtered and displayed on the surface at runtime. [7] 35

4.12 An example of hatching with dynamic solid textures. Note that stroke lines have the same size no matter how far away the surface is 36

4.13 A closeup of some tree models, rendered with basic hatch shading and contour outlines. 37

5.1 Various terrain models & camera angles that were used to compare performance across multiple selected styles. 39

5.2 A cartoonish style with a custom skybox. 42

5.3 A black and white sketch-like style with a closeup on some trees. . 43

5.4 A black and white sketch-like style with a wide-angle shot. This version showcases hybrid-overlay strokes with a textures stroke mesh. 43

5.5 A style using terrain textures with modified contrast channels. . . . 44

5.6 An orange style using heavy cell-shading, overlay strokes, and an overlay border. 44

5.7 A watercolor-like style created by warping texture coordinates. . . . 45

5.8 User study results regarding general aesthetic of the black-and-white style. Users were prompted with: "Please evaluate the scene demonstrated in this video on the following qualities" 46

5.9 User study results regarding general aesthetic of the green-terrain style. Users were prompted with: "Please evaluate the scene demonstrated in this video on the following qualities" 47

5.10 User study results regarding general aesthetic of the orange style. Users were prompted with: "Please evaluate the scene demonstrated in this video on the following qualities" 48

5.11 User study results regarding how well the black-and-white style achieved a "hand-drawn" look. Users were prompted with: "How well do you think this scene simulates a hand-drawn or 2D art style?" 49

5.12 User study results regarding how well the green-terrain style achieved a "hand-drawn" look. Users were prompted with: "How well do you think this scene simulates a hand-drawn or 2D art style?" 50

5.13 User study results regarding how well the orange style achieved a "hand-drawn" look. Users were prompted with: "How well do you think this scene simulates a hand-drawn or 2D art style?" 51

Chapter 1

INTRODUCTION

1.1 Photorealism vs Non-photorealism

The postmodern era has brought an explosion of creativity and originality to many aspects of our world. In particular, entertainment has been subjected to numerous revolutions in creative vision over the last few decades, and the rise of digital entertainment has provided more outlets for many to display their individuality and imagination through the use of computer graphics. Modern cinema, animation, and video games seek to pull their audiences in with unique and eye-catching visuals, and this trend has prompted the advancement of multiple new fields of graphics technology. Specifically, graphical rendering and image synthesis have expanded to encompass increasingly realistic and complex scenes. Photorealism in rendering has been of interest to many industries because of its potential to improve immersion and create lifelike environments. However, non-photorealistic rendering, sometimes called stylized rendering, has also flourished in modern entertainment due to its unique creative applications. In contrast to photorealistic rendering, non-photorealistic rendering prioritizes simplicity or artistic style over physical realism. Stylized rendering is often used to emulate human-drawn art styles in a 3D environment; this is sometimes referred to as stroke-based rendering. Figure 1.1 shows one example of a stylized rendition of a real-world environment.

(a) Actual photograph of cradle mountain.

(b) Stylized / NPR rendition of cradle mountain.

Figure 1.1: A side-by-side example of a photorealistic image (in this case an actual photograph) and stylized rendition of the same image. Stylized images generally forgo physical accuracy for improved visual appeal or style. Images obtained under creative commons from http://bjornfree.com/galleries.html

1.2 Modern Stylized Rendering

Before the advent of 3D animation, studios such as Disney and DreamWorks would draw 2D animation frames by hand, with limited computational support. Since then, many of these studios have shifted to 3D graphics pipelines due to their simplicity and reduced production time. Nonetheless, there is still significant demand for 2D animation in entertainment, both in cinema and for interactive applications. As a result, some research in this field has been conducted to explore options for combining 3D and 2D graphics pipelines.

Some of the obstacles to this technology involve visual consistency and performance constraints. As such, real-time applications like video games generally find it more difficult to utilize advanced graphical techniques in this field. Yet, there is still a large market for stylized art in games. Some examples of games which have received widespread acclaim for their stylized art include comic-inspired games like the

"Borderlands" series, cartoon-inspired games like "The Legend of Zelda: Breath of the Wild", and watercolor-styled games like "Ōkami". Many of these games feature open-world environments, which can introduce unique constraints and limitations due to the performance impact of large-scale 3D scenes. Artists are often restricted in the level of detail they can apply to game assets, to avoid hurting performance and player immersion. As a result, there is a need for advanced graphical systems which can provide artists with more creative freedom in game-development, while maintaining high performance on modern commercial hardware.

1.3 Contributions

The system introduced in this paper attempts to address the challenge of real-time stylized rendering in a large-scale 3D environment, by combining existing methods with specific adjustments that improve usability and performance. The contributions of this paper include implementations of hatch-line shading, contour & silhouette rendering, and various additional features. Specifically, we implement a hatch-line shading scheme that makes use of dynamic solids to improve texture coherence at varying depth levels, based off of research by Praun et al. and Benard et al. [29, 7] In addition, we combine this technology with a stroke rendering algorithm that extracts relevant feature lines from terrain and renders them with user-specified stroke textures. These methods are applied to multiple 3D terrain meshes, with various tree models spread throughout to provide detail commonly found in modern games. Notably, this paper significantly improves upon existing methods' performance in real-time, and allows for some unique customization options. We also provide a simplified application with an overlay GUI to demonstrate the usability of our system.

Chapter 2

BACKGROUND

Rendering large-scale scenes in real-time can prove to be difficult due to the large amount of scene data that needs to be processed every frame. High-density terrain models around 100,000 vertices, can be especially challenging to render accurately, because of their variable yet continuous geometric structure. In addition, as distance from the viewpoint increases, the amount of visible geometry often increases exponentially, which can result in severe performance degradation as the GPU struggles to process the large amount of data. As a result of these constraints, several standardized methods have been introduced in modern rendering systems to improve performance and visual quality in large-scale scenes. [4, 10, 21, 29, 34] Many of these methods can apply to both photorealistic and non-photorealistic rendering, although our focus is primarily on their relevance to stylized techniques. In this section, we give some background on how large scenes are rendered, and describe some high-level strategies for managing level-of-detail and rendering performance in large scenes.

2.1 World-Space Versus Screen-Space

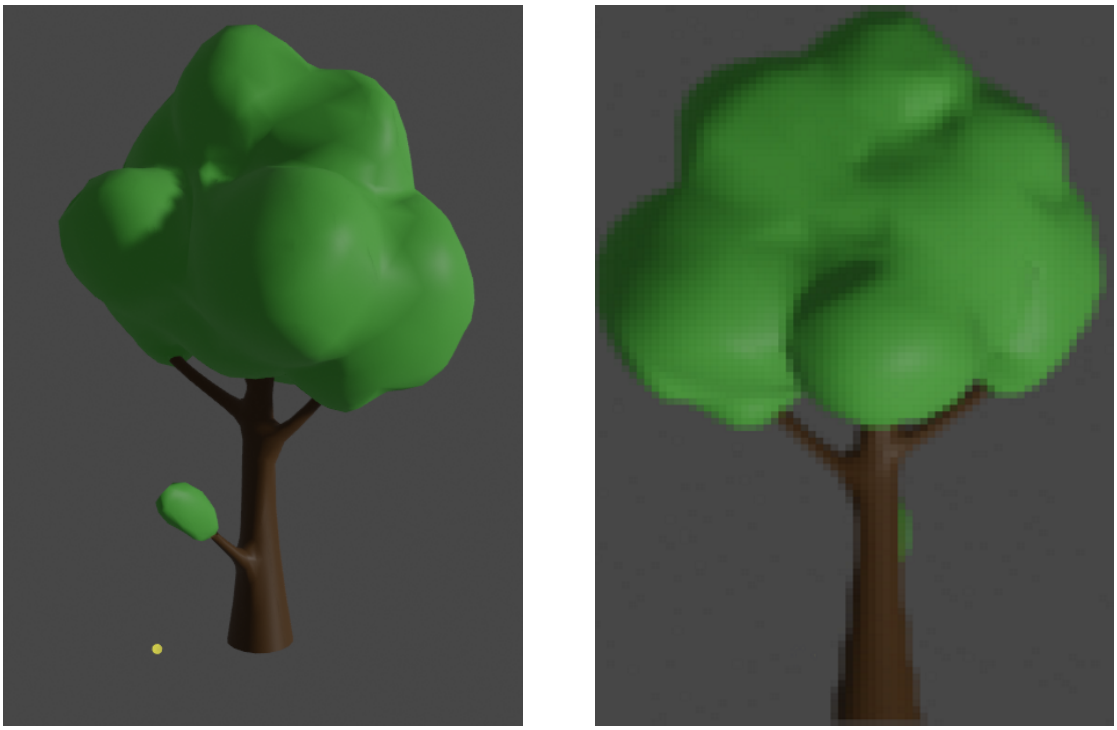
In terms of rendering hand-drawn styles, much of the challenge comes from blending techniques that operate in world-space and techniques that operate in screen-space. World-space is generally described as the 3D coordinate space in which scene objects exist. Positions in world-space consist of and x, y, and z coordinates. Conversely, screen-space is the 2D coordinate space defined by pixels on the screen, or in a frame. This space has a limited size, ranging from the lowest-leftmost pixel in a frame to

the highest-rightmost pixel, and many shaders operate in this space on a per-pixel basis. When targeting a hand-drawn aesthetic, operating in screen-space can often be beneficial because it aligns with the way human artists typically draw or paint on a flat surface. This introduces a need for techniques that can effectively translate between world-space objects and screen-space visuals.

2.2 Level-of-Detail Techniques

A common policy for large-scale rendering is to manage visual quality of the scene using varying levels of detail. As humans, we notice detail in objects that are closer to us, and tend to ignore details that are farther away. Similarly in rendering, we can artificially reduce geometric density for objects which are farther away from the primary viewpoint, without sacrificing the visual quality of the scene. This usually takes the form of "object proxies" or "billboarding", which reduces vertex density of objects or utilizes prerendered images of the objects, respectively, beyond a certain view distance. This reduces computational cost for the majority of objects in a large environment, and can even improve visual quality in some instances. An example of billboarding applied to tree objects is demonstrated in Figure 2.1

Additionally, standard methods for surface texturing can be impacted by surfaces which extend far into the distance. As geometry gets further from the camera, the number of pixels available to represent a texture decreases, which can result in pixelated or incoherent imagery at long range. A common solution is to use mipmapping, which involves generating smaller versions of the specified textures via sub-sampling techniques. [19] These "mipmaps" provide better quality textures for far-away surfaces, and are pre-calculated at load-time so they do not impact frame rate. Most modern rendering APIs, including OpenGL, have built-in methods for automatic



(a) Full 3D model of a tree.

(b) Captured 2D image of a tree, used for billboarding.

Figure 2.1: An example of a billboard used to reduce model complexity in large-scale scenes. The image in *b* replaces instances of trees that are far away from the camera. The reduced quality is not as noticeable at long range.

mipmapping, though usually custom mipmaps will provide more consistent visuals. When it comes to texturing discrete features, mipmaps are sometimes insufficient, and adjustments must be made. This is discussed in Section 3.3, with respect to hatch-line texturing.

2.3 Compute Shaders

The GPU is a powerful resource that enables rendering processes to be efficiently parallelized for performance. However, rendering processes are not the only tasks that can make use of the GPU. Compute shaders provide a framework for sending arbitrarily-defined data to the GPU for parallel processing, and retrieving it on the CPU after completion. In this way, compute shaders allow users to utilize the power of the GPU, without directly going through the standard render pipeline. These shaders can be extremely useful for parallelizing tasks such as complex feature extraction and non-uniform vertex manipulation. Given the prominence of GPUs in modern commercial hardware, making use of parallelization tools like compute shaders is key to improving system performance. In this paper, we utilize OpenGL's compute shader implementation to perform such tasks at real-time rates. Most notably, we use compute shaders to parallelize the extraction stage of one of our stroke rendering algorithms. This is expanded upon in Section 4.2.

2.4 Terrain Features

When it comes to stylized rendering, certain aspects of a 3D environment hold more relevance than others. When artists draw a landscape picture, they often omit small details like grass or tree leaves, and focus on capturing more prominent features like key object outlines or ridges. In an attempt to replicate these tendencies, stylized

rendering often involves defining and extracting specific features that best represent the most visually important aspects of the terrain. Some of these features include silhouettes, ridges, valleys, and contours. An example of some terrain features applied to a 3D scene is shown in Figure 2.2. Silhouettes are defined by the boundary between an object and its background; in other words, its outline. Ridges and valleys are often used in topological formats to distinguish regions where a surface is at a minimum or maximum height. They provide extra detail for functional purposes, but are not always the most visually appealing, at least in an artistic sense. Contours are defined by regions on a surface that are nearly perpendicular to the view direction. They are similar to silhouettes, but also include regions where parts of the surface are occluded by itself. Contours are one of the more popular features used to represent stroke lines, and have been expanded upon in various ways to include other surface details. One such expansion is "suggestive contours", which are described in Section 3.1. In this paper, we mainly focus on contours and suggestive contours, since they usually produce the best results for achieving hand-drawn art styles.

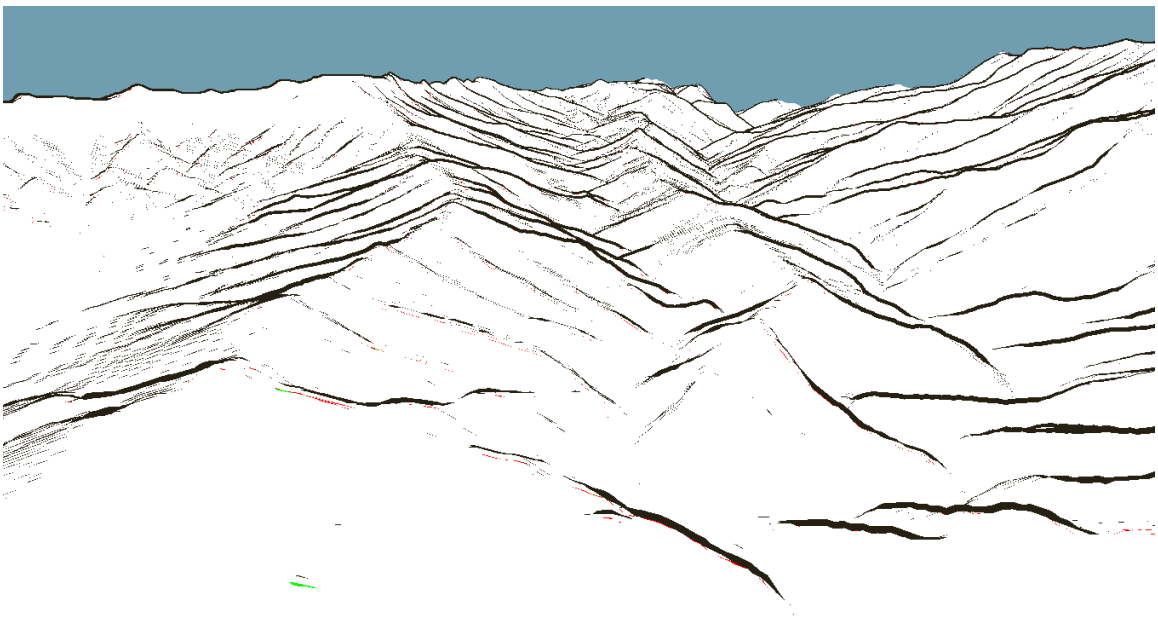


Figure 2.2: An early iteration of our terrain model, with contours rendered as black lines.

Chapter 3

RELATED WORKS

The goal of stylized rendering is primarily to achieve a 2D hand-drawn look for a scene, using standard 3D models as input. The visual quality mainly depends on how efficiently a method can extract and convert relevant 3D geometric data to a 2D screen-space format, and match the form that a human artist might apply to an image. Certain aspects or features of a 3D scene typically hold more relevance in 2D than others for recognizing shape and style, including contours, silhouettes, suggestive contours, and shaded regions. This section will address some existing methods for extracting these features, and representing them in a two-dimensional format. Research on this topic ranges from specific practical applications to more theoretical approaches.

3.1 Suggestive Contours

One of the earliest techniques for representing shape in 2D involves the extraction of "contours", which are defined mathematically as regions on a 3D model where the surface normal is perpendicular to the viewing direction. Contours provide a solid representation of object outlines, beyond a simple silhouette, though they cannot provide detailed visuals for more complex surface features. The use of contour features has been extended by DeCarlo et al. [9], who define the term 'suggestive contours' as regions of a surface that are "almost contours". To summarize, suggestive contours are regions that would be considered contours in an adjacent viewpoint, and they provide much a more detailed visual representation of a surface for human perception. An

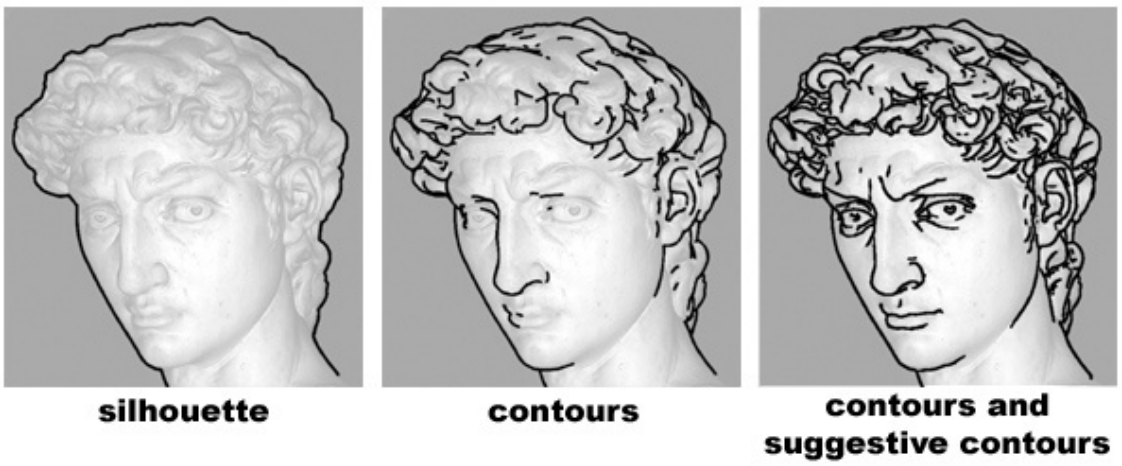


Figure 3.1: An example mesh with highlighted edge features. Note the improvement in detail when adding contours, and suggestive contours. [9]

example of these features is demonstrated in Figure 3.1. Contour/suggestive contour information can be calculated on the CPU as a precomputation phase, and then used for stylization while rendering on the GPU. Many stylization methods stem from this topic, and explore ways to work with silhouette outlines, contours, and suggestive contours. Some noteworthy examples include [28], [10], and [23], all of which explore ways to render edge features similar to or based on surface contours. Also worth noting is research by Xu and Chen [34], which generates point-based geometry from scanned environments. Their method extracts features from a point-based environment with a unique set of classification algorithms, and it is distinct from the usual vertex-edge contour approach.

3.2 Stroke Rendering

No matter which terrain features are used, extraction is only the first step. Many papers have explored rendering methods for specific edge features, in an attempt to create longer "stroke" lines which produce a more human-made look. Northrup et al. [27] demonstrate a hybrid approach to outlining silhouettes, adding a few more

steps to earlier methods which extract contour 'edges' and construct brush strokes from them. This method makes use of shaders to render encoded reference data efficiently, but relies on some CPU-side involvement to retrieve relevant edges from the geometry. While a bit less efficient, the hybrid approach is one of the more flexible and extendable methods for determining edges for stroke generation. Further research, such as that of Kalnins et al. [14], has largely focused on improving methods like the hybrid approach, to provide inter-frame coherence by preserving key feature points across many frames. These improvements also inherently improve performance by retaining calculated data from previous frames, rather than constantly recomputing.

Aside from purely screen-space alternatives, the construction of strokes and their placement in 3D space has been a subject of interest in stylized rendering as well. Some methods take an artist-focused approach, like that of Kalnins et al. [15], who implement a method for interactively applying strokes directly to the surface of 3D models. Methods like these focus on maintaining coherence across the surface of objects, but rely on users to place strokes and textures accurately. Other methods including [20], [27], and [18] are generally more concerned with synthesizing brush strokes in screen-space, after relevant features and contours have been extracted. These techniques are less reliant on user input, and can be customized by revealing specific parameters for editing. An excellent survey of modern surface-based stylized rendering techniques is provided by Lawonn et al. [22].

3.3 Painterly Rendering

One popular method for obtaining artistic frames is called "painterly rendering". This method is originally described by Barbara J. Meier for Walt Disney Animation [26], and involves painting several brush strokes across a frame, with colors and directions

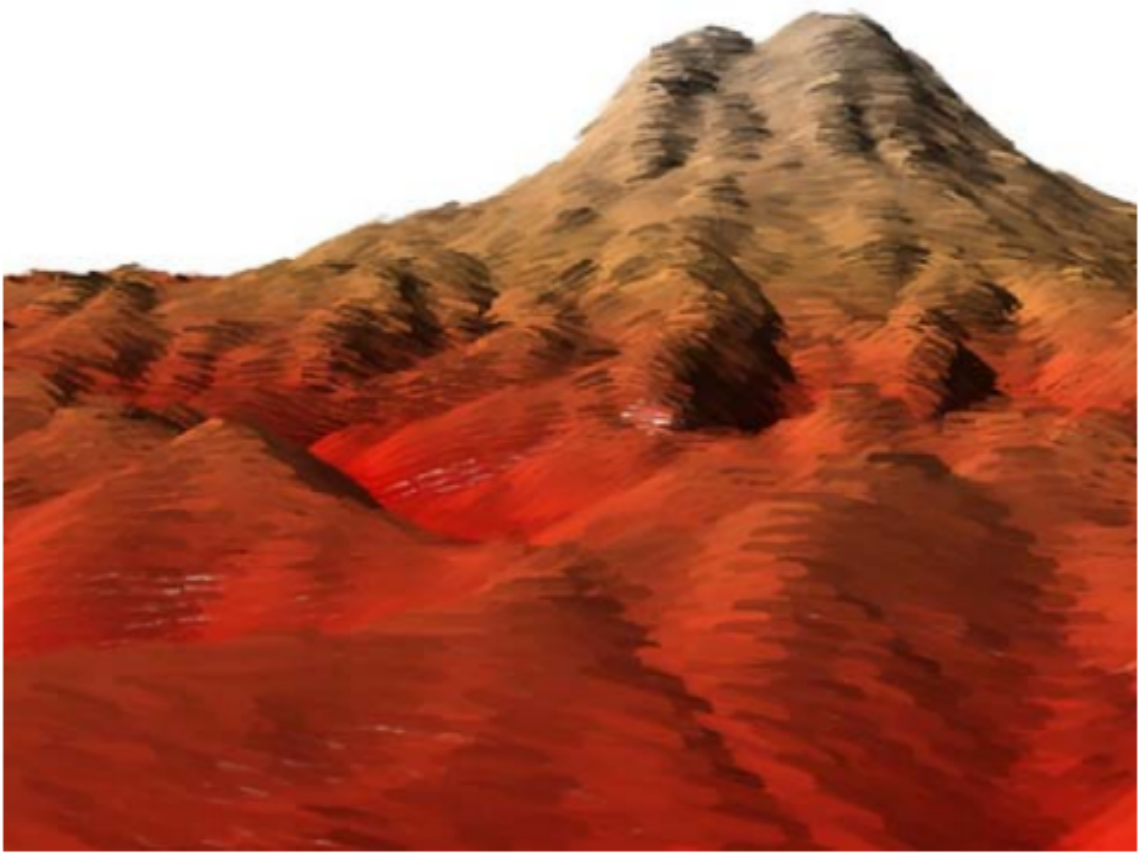


Figure 3.2: A model of Mt. Rainer, rendered using a painterly style. [4]

arranged according to scene geometry. Painterly rendering simulates human drawing patterns somewhat, by placing individual brush strokes to form an abstract representation of a scene. Further research, including [11] and [4], has extended this method to allow for more user-customization and different scene types, including terrain environments. Unfortunately, painterly rendering is not as generalizable to styles which rely on few discrete stroke elements, like pencil sketches or ink-silhouette painting. In this paper, we do not focus on painterly methods specifically, but many of the ways in which painterly methods represent strokes are applicable nonetheless. An example of terrain rendered with a painterly technique is shown in Figure 3.2.

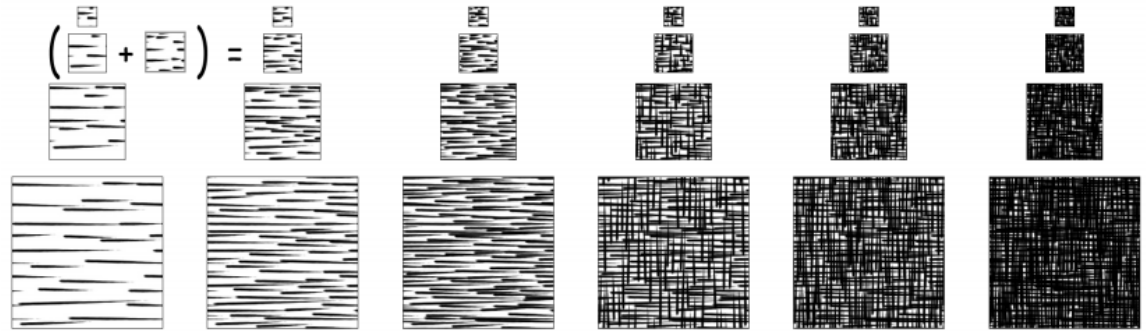


Figure 3.3: An example of real-time hatch textures, generated by overlaying lighter tones on top of themselves at varying intervals. [29]

3.4 Stylized Texturing

Another technique commonly used to achieve stylized 2D output is the use of surface texturing methods. Textures can be provided in many forms, and can directly contain hand-drawn artifacts which contribute to stylization, if mapped properly. As an early example, Praun et al. [29] utilize a specialized texturing method to apply cross-hatching to shaded regions. This method generates hatching textures with some variation, and blends corresponding densities to essentially 'stack' the hatch marks in regions that should be darker. This hatching method is demonstrated in Figure 3.3. Unfortunately, the problem of mapping the texture still exists, and basic mipmapping can sometimes cause issues for certain textures and models. Benard et al. [7] directly address this problem by using frequency adjustments to dynamically scale and shift textures depending on relation to the scene camera. In short, the method exploits humans' inability to perceive certain visual shifts to maintain texture coherence at any distance. In this paper we combine these two technologies to achieve real-time hatching that dynamic sizes itself based on view distance. This is discussed in detail in Section 4.4.

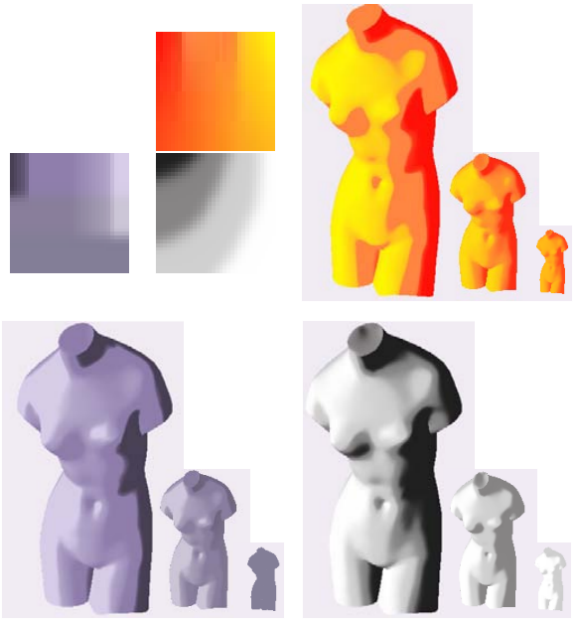


Figure 3.4: An example of toon shading, applied with various toon textures, defined in [2].

3.5 Color & Shading

In addition to producing discrete elements for stylization, like brush strokes or hatch lines, continuous shading and colorization are important for maintaining coherence and creative freedom. Countless research papers have discussed shading methods in detail, though the subjective subject matter can make it difficult to produce measurable results. Research by Sloan et al. and Bousseau et al. discuss shading methods relevant to hand-drawn art styles. [31, 5] These include methods for illumination extraction and morphology comprehension to achieve such styles as lavis watercolor and oil painting. Another commonplace technique is to utilize cell-shading or "toon shading" to produce cartoonish styles. Barla et al. [2] provide an excellent example of this, shown in Figure 3.4 with extensions for multi-color gradients and backlighting.

Chapter 4

IMPLEMENTATION

In this chapter, we present our system for rendering large-scale terrain environments in a stylized manner, specifically targeting pen and ink renderings. To implement our system, we use the OpenGL API alongside a variety of helper libraries including trimesh, GLFW, and imgui. [1, 8, 30] The final product makes use of standard GPU functionality for rendering and computation.

Our system includes algorithms for stroke-based rendering of edge features, surface hatching, and tree rendering. Stroke rendering refers to the representation of discrete brush or pen strokes across certain key areas of the terrain. Strokes can improve the hand-drawn aesthetic, since they produce similar visuals to what artists create when drawing or painting in 2D. To accomplish this effect, we implement two different methods, each with its own benefits and limitations. For sake of simplicity, we call these two methods *curvature-based strokes* and *hybrid overlay strokes*. Each of these techniques are demonstrated by the final system, and provide different style options for users to select from.

In addition to stroke rendering, we utilize texture-based hatch-line shading to provide additional detail to the scene, and improve the hand-drawn look of rendered frames. Hatching replaces smooth surface tones with discrete lines, which makes it look like someone drew directly onto each surface to represent shaded regions. Finally, we render tree objects across the terrain with simplified versions of the aforementioned technologies for stroke and hatch rendering. In the following sections we outline each of these technologies and summarize their contribution to the final product.

4.1 Curvature-based Strokes

The first type of stroke rendering method relies on precomputed curvature values for the terrain mesh. Specifically, we precompute principle directions, curvature, and the derivative of curvature for the entire visible surface at load time. This process is automatically performed by the trimesh library, following the algorithms defined in [9]. This curvature information can be analyzed on the GPU to determine if a given fragment is a contour or suggestive contour, for any renderable point on the terrain mesh. For both contours and suggestive contours, a 'feature size' parameter can be provided to modulate the threshold between contour and non-contour points. Rendering either of these stroke types can be done by simply coloring the contour and suggestive contour fragments a solid color, but this is fairly restrictive to users in terms of creative freedom. Ideally, we would like to be able to parameterize strokes in some way, so that users can customize contour lines length-wise or apply textures.

4.1.1 Parameterizing Orientation

To accomplish this, we introduce a way to estimate directionality and stroke-width for contours entirely within a fragment shader. For a given fragment, the available surface normal is converted to screen-space using available transform matrices. Since contours usually occur in regions where the normal is perpendicular to the view direction, we can use the screen-space normal to approximate the direction of a contour stroke. A perpendicular direction vector is obtained by performing the cross product between this normal and the camera direction. This direction vector can then define a texture coordinate space along the stroke, where the 'U' coordinate follows the primary vector direction, and the 'V' coordinate follows the screen-space normal

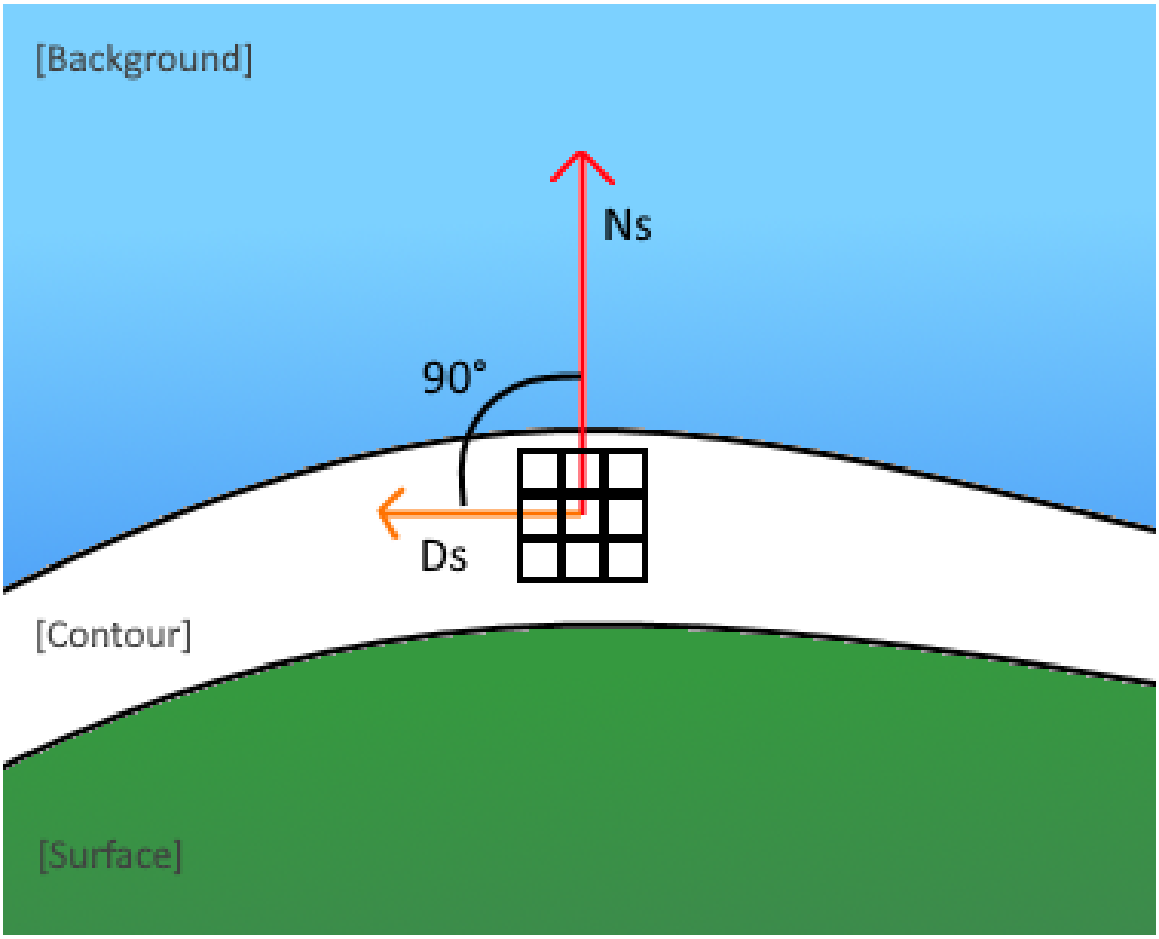


Figure 4.1: Curvature-based strokes utilize the screen-space fragment normal (Ns) and its perpendicular counterpart (Ds) to approximate stroke directionality.

direction. An example of this is shown in Figure 4.2. In this way, we parameterize the "stroke-space" in both the length and width dimensions.

4.1.2 Stylization Options

Using arbitrary scaling factors for each dimension, a user-generated texture can then be applied to provide more customizability to contour strokes. This method is only an estimation of a true texture-space, and is distinctively non-linear, though in most cases it comes close enough to be reasonably coherent. This can be seen in Figure 4.3.

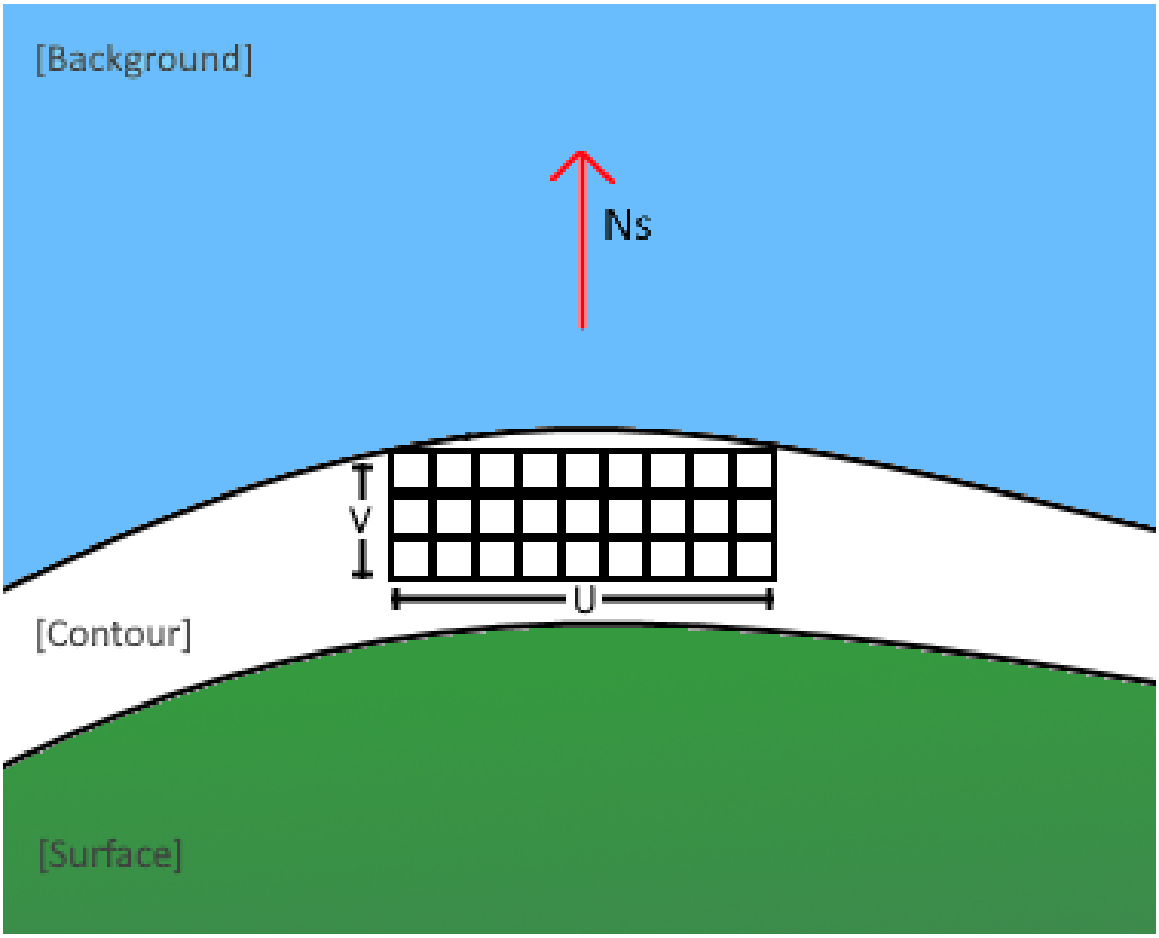


Figure 4.2: Once the directionality of a stroke has been determined, the space can be roughly parameterized along the length and width of a perceived stroke. In this example, U represents the horizontal stroke dimension, and V represents the vertical stroke dimension.

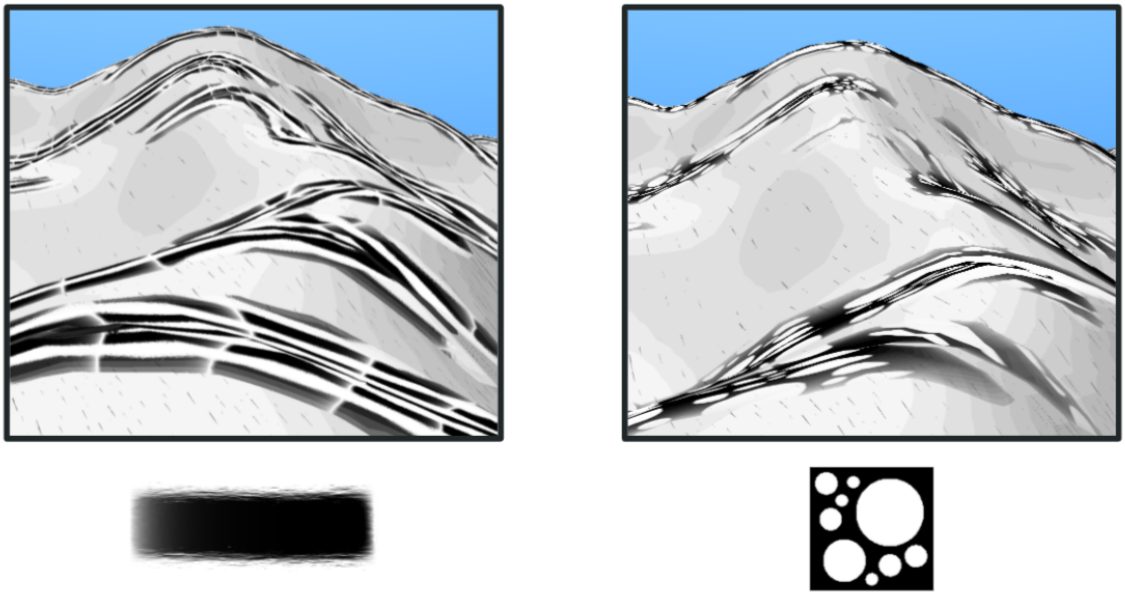


Figure 4.3: Two examples of stylized curvature-based strokes. The textures beneath each image are the same used in the images above them. While some warping is visible, the textures mostly face the direction of the perceived strokes.

Other stylization aside from textures can be used as well, including alpha tapering along the width, or periodic warping. While this method is spatially and temporally coherent, the representation does not provide any information on stroke endpoints, since it only approximates directionality on a per-fragment basis. As a result, this method is limited to the use of stroke textures that are tileable in both directions, since obtaining an accurate linear representation of stroke length and width is difficult without more information. In addition, suggestive contours cannot use this method since their directionality does not correlate with surface normals in the same way. An example scene using curvature-based strokes is shown in Figure 4.4.

Figure 4.4: Contour strokes are rendered in this scene as solid black regions, with slight alpha tapering along the edges. Curvature-based strokes are spatially and temporally coherent, but are limited in what options they provide for stylization.

4.2 Hybrid Overlay Strokes

Curvature-based strokes provide a solid baseline for representing coherent edge features, but they still lack customizability in some areas. We don't have an understanding of stroke endpoints and the rough parameterization is only suitable for tilable textures and approximating directionality. The second type of stroke rendering implemented in this paper resolves these issues by providing a fully linearly-parameterized, texturable stroke mesh to work with. However, it is a much more involved process with a higher performance impact. This method includes a hybrid approach that combines various screen-space and world-space operations to properly orient and display discrete stroke lines in the scene. This process involves three steps:

- Feature Extraction
- Edge Chaining
- Rendering and Stylization

In summary, contour edges are extracted from the scene via multiple shader passes, then connected together to form coherent chains, and finally rendered to the screen with various stylizations. In the following sections, we describe each of these stages in detail, along with our methodology for improving temporal coherence.

4.2.1 ID Reference Image

Feature extraction is the first stage of the hybrid overlay stroke system, and involves finding specific edges and contours of the visible surface to pass down the pipeline. To do this, we extend [20] and [27] to utilize the concept of an ID reference image.

Specifically, we render each "contour-edge" as a single line segment with a color that uniquely identifies the edge. For our implementation, we represent the ID using a combination of the edge's two end-point positions, encoded into a 32-bit RGBA value. Since no two edges share exactly the same two endpoints, every ID is unique. In this context, a "contour-edge" is defined as any visible edge which joins a front-facing edge with a back-facing edge. This pass is performed using a geometry shader, with a pre-rendered depth map used to enforce occlusion, so only visible portions of an edge are rendered. In addition, this reference image can be rendered at a lower resolution than the other passes without losing much accuracy. We obtain a significant performance increase for large-scale scenes by reducing the resolution of the ID image to 480x270 pixels. For a full rendered scene at 1920x1080 pixels, using a 480x270 ID image reduces computational complexity for this stage by a factor of 16, and can result in a framerate increase anywhere between 10 to 30 FPS depending on the scene. An example ID image mock-up can be seen in Figure 4.5.

4.2.2 Feature Extraction

Rendering the ID reference image allows each contour edge to be converted into screen-space via the rendering process. Next, each edge must be extracted from this ID image into a usable list of edges. This task can be extremely slow sequentially, so we use an OpenGL compute shader to parallelize the process. For each pixel in the scene that has a color value greater than zero, we encode the ID as a 32-bit integer and append it to a list of edges. This list gets passed to the CPU afterwards, and each non-duplicate edge is decoded and prepared for the next stage. At the end of this extraction stage we have a list of edges, represented by their maximum and minimum screen-space positions. We could render each edge as its own stroke line,

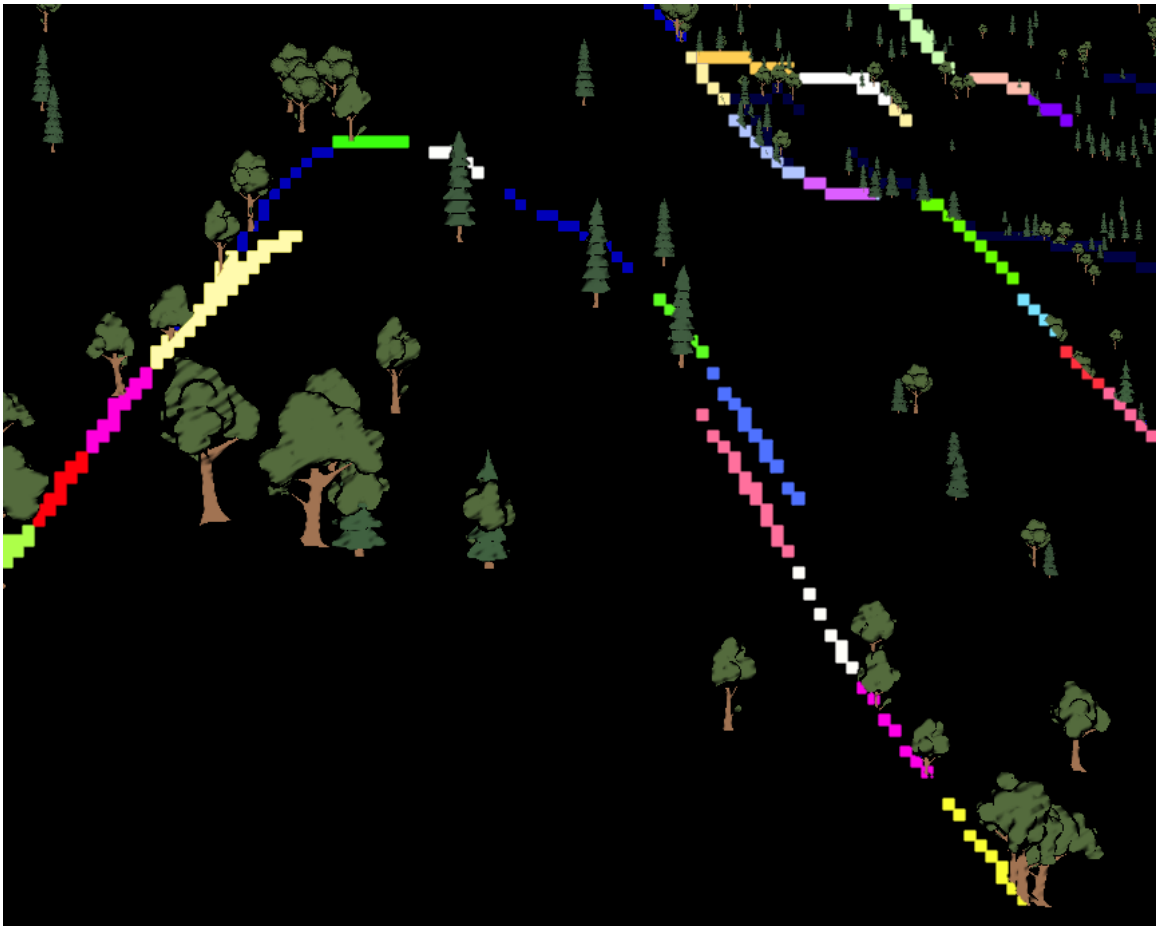


Figure 4.5: A sample from a 480x270 ID reference image, used for edge extraction. Each edge is rendered as a unique RGBA value representing its ID. Colors have been exaggerated for demonstration purposes.

but in practice this results in poor visual coherence, so we employ a complex chaining algorithm, described in the next section.

4.2.3 Chaining Algorithm

Rendering each edge as its own stroke produces a bunch of small inconsistent segments. Edges must be joined together end-to-end in order to form longer chains that are coherent across all ranges. For this, we implement a chaining algorithm which selects candidate edges that meet some criteria to be chained together in a doubly-linked list format. We traverse the entire list of available edges, and scan nearby pixel locations for each edge to find potential chaining candidates. Specifically, we select a neighboring edge for chaining if it is within some specified distance D, has an angular difference less than A, is not overlapping the selected edge, and has not already been joined to another edge. If these criteria are met by multiple neighbors, we select the neighboring edge that minimizes the distance and angle difference. For our final implementation, we select a value of 4.0 pixels for D and 60 degrees for A. The diagram in Figure 4.6 demonstrates this process. Below is a pseudo-code implementation of the candidate selection process.

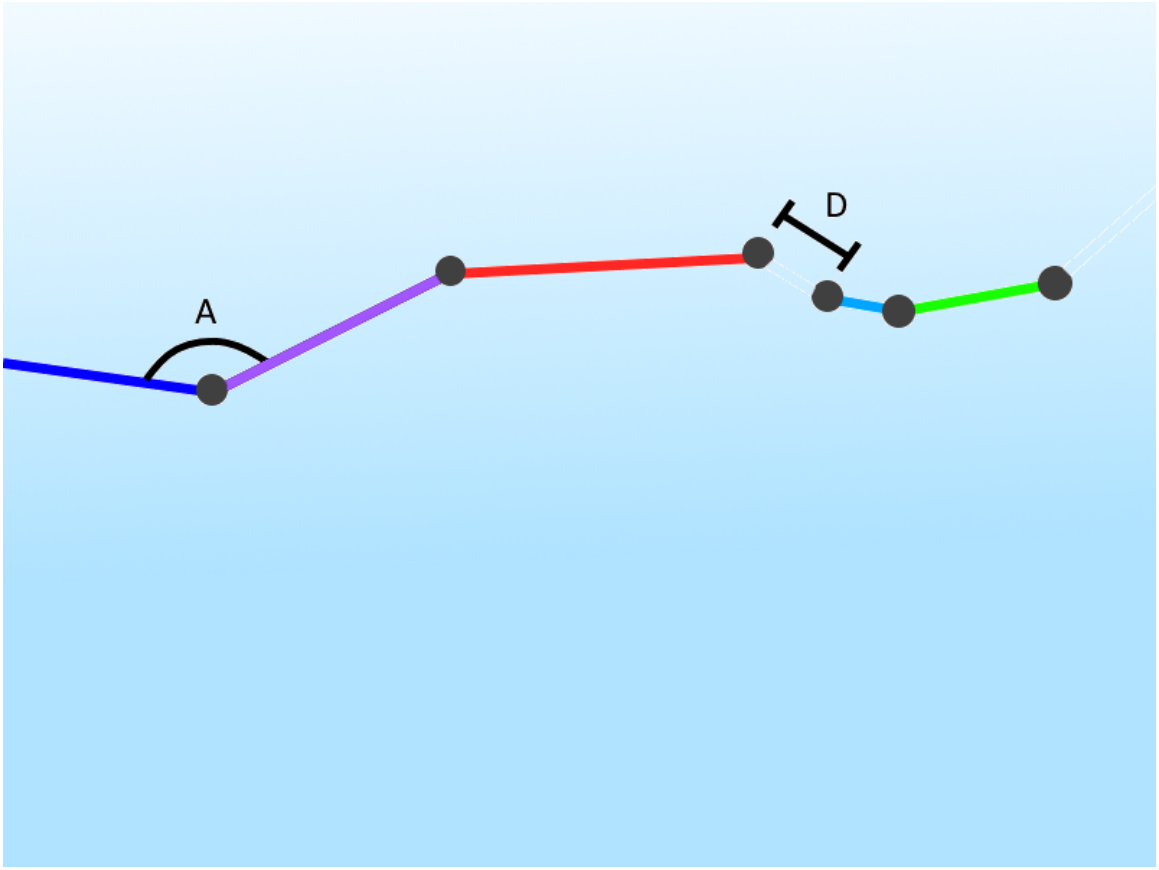


Figure 4.6: A diagram depicting chaining constraints. A denotes a user-specified angle constraint for chaining, and D denotes a user-specified distance constraint in pixels.

```
for each edge E in list of edges
{
  for each neighboring edge N nearby
  {
    if ( distance(E,N) < D &&
         angle(E,N) < A &&
         E & N not overlapping )
    {
      chain_edges(E,N)
} } }
```

Once a chaining candidate is chosen for a given edge, the two edges are linked together in a doubly-linked list structure, and the four endpoints (2 from each edge) are stored in a list based on the order of chaining. Each generated chain is appended to a list of all chains, which gets passed to the draw call at render time. If one edge is already part of a chain, the other will simply be linked to the existing structure, and its two endpoints will be added. If both edges are part of a chain, the two chain structures will be merged into one, replacing the existing chains. Finally, once all edges have been matched with any suitable chaining candidates, any edge which has found no partners is converted to a standalone chain and added to the list. At the end of this stage, we have converted all extracted contour edges to longer chains which provide a more stroke-like appearance.

4.2.4 Stroke Mesh Construction

The input to the draw call for stroke mesh objects is a list of chains, which each contain their own list of segment endpoints. In practice, each of these vertices generally define one vertex of the stroke mesh. However, we need to provide the vertex chain with

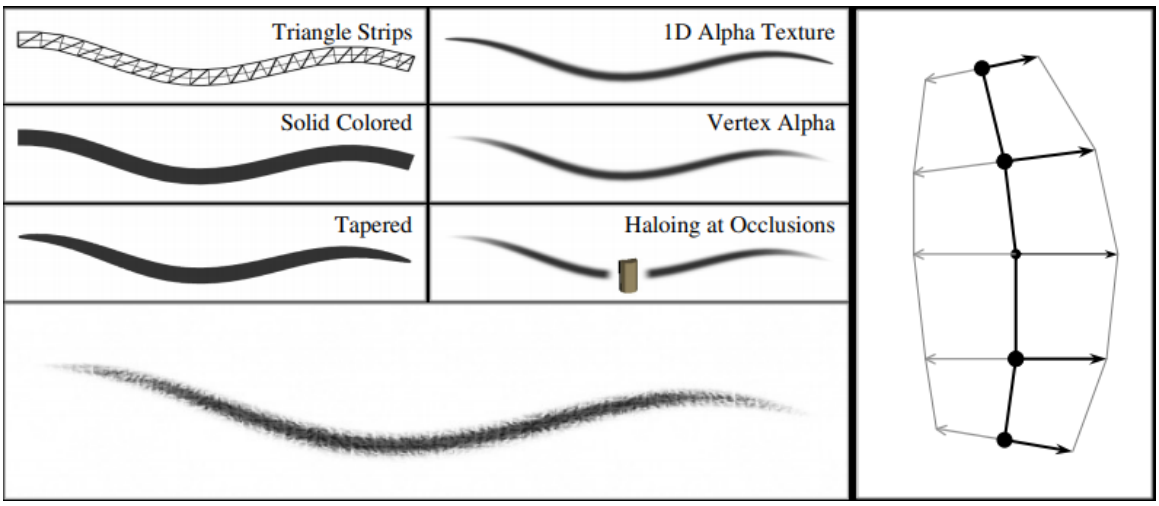


Figure 4.7: An example of how triangle strips can be formed from the list of vertices for a given chain (right). The other images demonstrate some example stylizations that can be applied to the strips using a shader. [27]

some width to be able to render it with stylizations. To do this, we apply a few modifications to the structure and smooth out the transitions between edges.

First, we define a stroke direction for each vertex in the list, as the difference between the current vertex in the chain and the next. Triangle strips are then formed by extending each vertex outward along the direction perpendicular to this stroke direction. In this way, two new vertices can be generated for each existing vertex in the chain, on either sides of the stroke. This expansion is shown in Figure 4.7. The width of each stroke mesh is defined by the user, and the length is defined by the sum of the distances between each vertex and its following counterpart in the chain. We also track the relative lengths of each segment in the stroke, to more evenly parameterize the surface lengthwise. As a final addition, we round off the endpoints of each stroke through the addition of six vertices; three at each end. The vertices are arranged in a triangle pattern, shown in Figure 4.8, to create a rounded edge for each stroke.

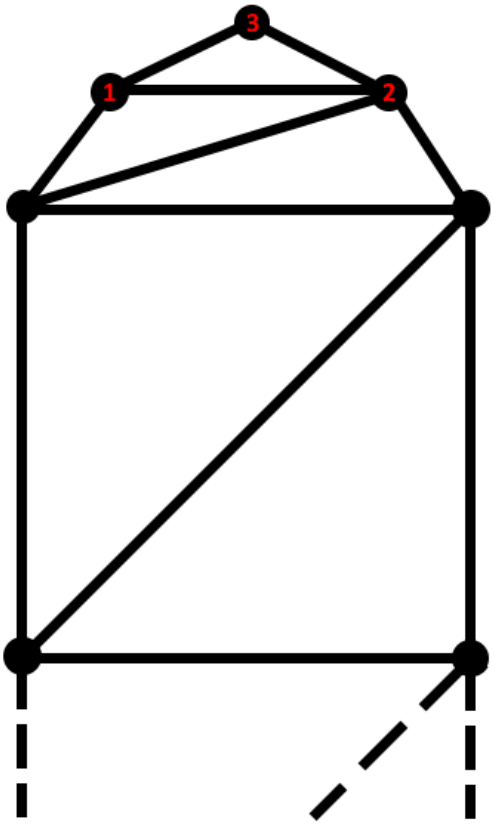


Figure 4.8: Three vertices are added to the ends of each stroke to create a rounded edge. The red labeled points indicate added vertices.

4.2.5 Stroke Rendering

Rendering the stroke mesh is fairly straightforward after it is appropriately generated. For simplicity, each pixel coordinate is converted to a world-space position, with a depth of zero to ensure that the stroke renders in front of all other objects in the scene. These stroke meshes exist in the same space as the terrain model itself, but are close enough to the camera that they will always appear as if they are overlayed on top of the scene, like brush strokes on the screen. Since the mesh surface is fully parameterized along the length and width, various shader techniques can be used to stylize the strokes from the GPU. In our implementation we demonstrate a simple alpha and width taper at the ends of each stroke, and apply pre-made stroke textures to add detail. This rendering method is heavily based off of [27], and some examples of possible stylizations are shown in Figure 4.7.

4.3 Temporal Coherence

Feature extraction occurs on a frame-by-frame basis, and features such as contour edges can often be present in one frame but absent or significantly altered in the next. This results in a considerable amount of visual noise when a user moves throughout the scene while hybrid overlay strokes are enabled. To address this, we implement a few techniques to help mask the noise, and smooth transitions between multiple frames. Further exploration on this topic is discussed in Section 6.

4.3.1 Motion Field

To help transition stroke vertices between frames, we implement a screen-space motion field that emulates optical flow. In our scene, the only significantly moving object

is the camera, so mapping ego-motion is sufficient. Trees have slight movement do to animation as well, but they do not utilize hybrid-overlay strokes, so temporal coherence is not a concern for them. This mapping is done by tracking view matrices from previous frames and using them to determine the change in position between pixels over time. Specifically, we determine a screen-space motion vector v for a given pixel by transforming the current world-space position at that pixel by both the current view matrix V and the old view matrix V_o. The two resulting vectors are then converted into screen-space and subtracted from each other to obtain v. An example depiction of a motion field is presented in Figure 4.9. For our system, we use the view matrix from ten frames prior for V_o, and we only perform a full stroke update every tenth frame. This does not completely remove visual noise, since there are new features being introduced and removed on update; however, it does reduce the frequency of flickering and other visual artifacts.

4.3.2 Adjustments to Smooth Motion Field

The motion field produced by this method is not perfect, especially when dealing with edges that become occluded or drastically change. For example, when the camera rotates too quickly, motion vectors have a tendency to overcompensate and have too great a magnitude. To help reduce this effect, our implementation detects camera rotation above a certain threshold and performs a full stroke update anytime it is too high. Thankfully, visual noise produced by the increased update frequency is largely unnoticeable in situations where the view is changing quickly. In addition, motion vector computations will produce incorrect values for pixels which do not contain the primary terrain surface (i.e. pixels which render the horizon/sky). We resolve this by supplying a large box mesh around the scene which provides world-space coordinates to the motion field process, but remains invisible during the final

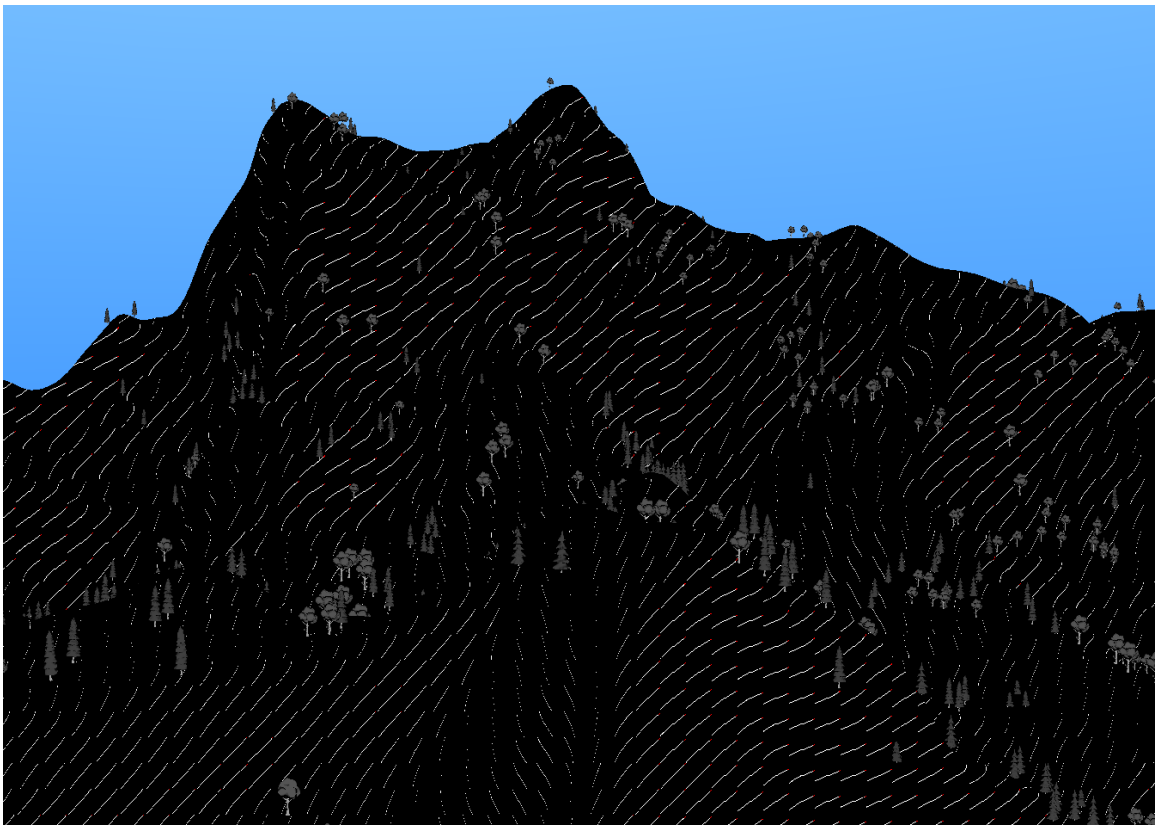


Figure 4.9: A depiction of a motion-field used to improve temporal coherence. Each white line representation the direction that the terrain has moved in the last ten frames

render pass. This addition helps keep silhouette edges from growing to infinity or disappearing completely.

4.4 Hatching

Aside from stroke rendering methods, surface texturing can play a large role in depicting stylized environments. A useful tool for achieving a hand-drawn look via line shading is a texturing technique called "hatching". The idea is to provide surface detail by applying hatch marks to a surface, much like how an artist might shade a region by crosshatching. The hatch lines can be used to replace or supplement continuous shading methods, providing a more natural look. The baseline for this technology stems from [29], and has been improved by adapting the Dynamic Solids technology from [7].

4.4.1 Dynamic Blending

Before running the program, multiple hatch textures can be provided by the user, each with a different hatch mark density. These textures can be produced using a separate program, user generated, or obtained by overlaying lighter textures on top of each other with some arbitrary offset. Textures meant to represent darker shades can be overlayed on top of each other more times to achieve a more dense distribution of lines. At runtime, the textures are dynamically blended together by interpolating between them; this is modulated by the diffuse light level, or "tone", at a given surface point. Using only the hatching textures for shading can create a jarring look, so the blended hatch textures are used in conjunction with traditional shading methods to smooth out the appearance of terrain surfaces.

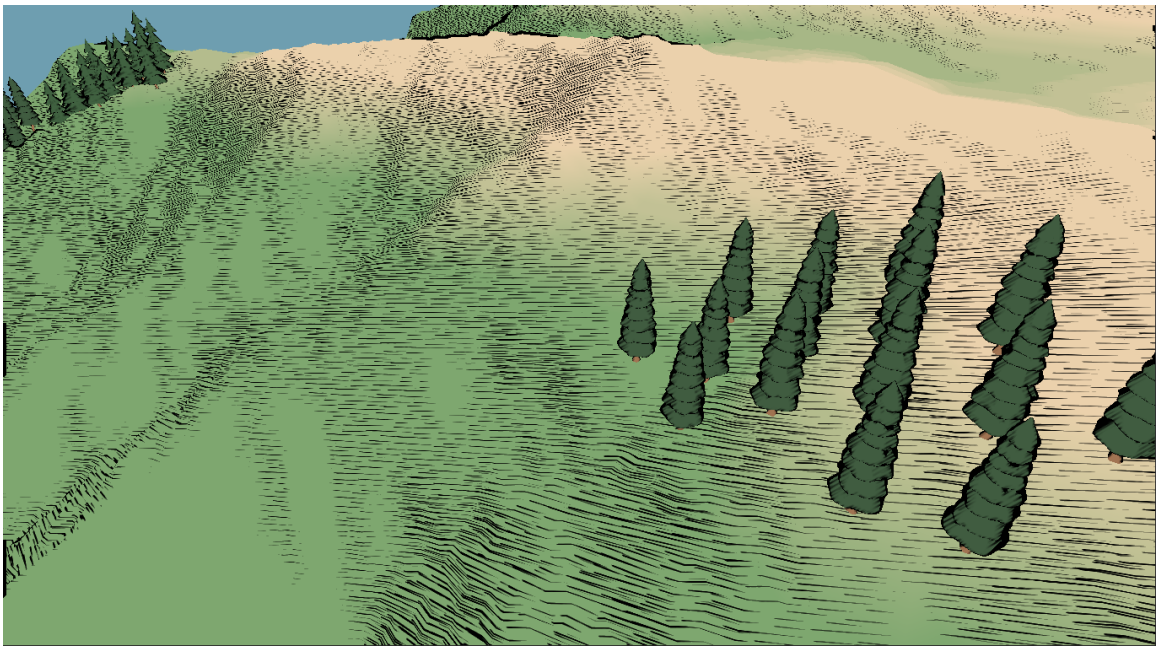


Figure 4.10: An example of hatching with basic mipmapping. Note that surfaces further from the camera become noisy and lose distinct stroke-line visuals

4.4.2 Dynamic Solid Textures

One issue that arises when using textures that display discrete elements, such as hatch lines, is a gradual loss of detail as scene depth increases. Smaller mipmap levels do not have enough space to represent distinct lines, so textures gradually fade to a solid color as the range increases. In addition, all stroke and hatch lines should ideally be of similar width, regardless of their distance to the camera, to more realistically portray a hand-drawn visual style. Traditional mipmaps are not sufficient for conserving these discrete visual elements, so we turn to another approach.

The Dynamic Solid method outlined in [7] provides a solution to all of these issues. In summary, the dynamic solids method utilizes 3D textures and specific frequency shifts to provide an "infinite zoom mechanism." Two-dimensional hatch textures can easily be pre-loaded into a 3D format by duplicating the image for each 'layer' in

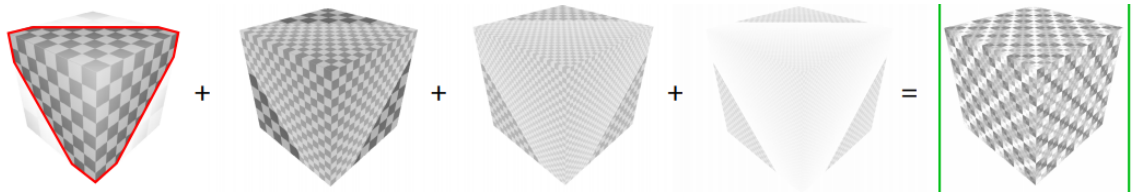


Figure 4.11: An example of a dynamic solid texture with a checkerboard pattern. Each consecutive image shows a higher level of texture scaling that can be modulated using specialized blending weights. The rightmost image shows the combined dynamic solid texture, to be filtered and displayed on the surface at runtime. [7]

the 3D structure. An example of how textures can be stored and blended using this method is shown in Figure 4.11. Then at runtime, the world space position (x, y, z) of a given surface point can be used to index into the dynamic solid texture using specialized blending weights to obtain the pixel color. This provides accurate, depth-modulated texture scaling that better approximates how hatch lines would look on a 2D surface. For the discrete hatch line textures, a binary filter is applied after using the dynamic solid method, to prevent the lines from blurring together. Examples of scenes with and without the dynamic solid implementation are shown in Figures 4.10 and 4.12, respectively.

4.5 Trees

Modern open-world games typically include many different types of objects in addition to standard terrain models. To more accurately portray a scene as it would be in modern entertainment, we render a number of procedurally placed tree models alongside our standard terrain mesh. These trees make use of some of the same technologies discussed in previous sections, while also utilizing a few specific optimizations. Specifically, the various tree models are rendered with solid black contour-based silhouettes and light hatch textures applied in shaded areas. Because the tree models are relatively small and have a medium-sized vertex count, we determined that basic

Figure 4.12: An example of hatching with dynamic solid textures. Note that stroke lines have the same size no matter how far away the surface is

contour outlines are the best option, both visually and with respect to performance. For similar reasons, only one shade of hatch marks are applied, based on the diffuse coefficient at each surface point. In order to include many trees in the scene, we employ a billboarding strategy to reduce model complexity at long range. This involves replacing tree meshes beyond a certain depth with a simple quad structure that always faces the camera and renders a precaptured image of the tree in question. We also provide basic animation to tree leaves, in the form of slight vertical & horizontal oscillation, to add some motion to the scene.

Figure 4.13: A closeup of some tree models, rendered with basic hatch shading and contour outlines.

Chapter 5

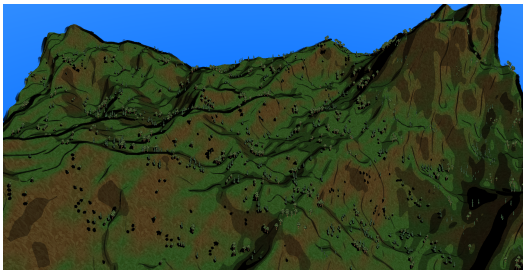
VALIDATION & RESULTS

This paper outlines a framework for stylized rending in real-time that combines multiple existing methods with some novel techniques. Our implementation provides improved performance and creative flexibility compared to previous methods, due to the adjustments described in previous sections. Additionally, we showcase multiple options for stroke rendering which provide varying degrees of customizability and performance impact. In this section, we describe our results in detail, including the various visual and performance-focused validation methods that we used.

5.1 Performance Results

A major focus of this project was to achieve stylized rendering in real-time, specifically for large-scale environments. As described in previous sections, multiple techniques were used to improve performance for scenes with a large vertex count, in order to maintain interactive frame rates. Previous papers on this topic vary significantly in terms of available GPU power and the hardware that is used for testing. Some of the papers that we use as a foundation, including [29] and [27], are much older and bottlenecked by the architectures of their time. For the most part, our implementation significantly improves upon the performance of previous methods, primarily via the use of compute shaders and adjustments to stroke and hatch rendering algorithms.

All of our testing was performed on medium-range commercial hardware, and our timing results were recorded using an AMD Ryzen 5 3600X CPU, and an NVIDIA GeForce 1050Ti GPU. We tested multiple styles and camera angles using both the



(a) Wide angle shot used for performance testing (shown with GN style).

(b) Low angle shot used for performance testing (shown with BW style).

Figure 5.1: Various terrain models & camera angles that were used to compare performance across multiple selected styles.

curvature-based and hybrid-overlay stroke rendering techniques, in order to evaluate the impact of each. Tables 5.1 and 5.2 show the average frame timings across multiple tested scenes, along with the corresponding frame rates. The black-and-white, green terrain, and orange styles are referred to here as "BW Style", "GN Style", and "OR Style", respectively. In addition, a baseline render is provided via a standard Phong shading implementation. Figure 5.1 shows some of the scenes used for testing.

As demonstrated by the data, our rendering system performs better when less terrain is visible. This is a common pattern in computer graphics due to view-frustum culling preventing non-visible data from being processed by the GPU, thus saving frame-time. However, it's likely that our implementation for stroke rendering exacerbates this problem. As the number of edges that are on-screen increases, the amount of work that needs to be performed by the chaining algorithm increases, especially for edges that are close to each other in screen space. As a result, wider-angle camera shots impact performance more heavily because more edges are visible and the edges are smaller and closer together on the screen. Curvature-based stroke methods do not have this restriction because they do not depend on the number of edges on-screen.

Each method was tested against three scenes, two with a 90,000 vertex model, and one with a 275,000 vertex model. The two scenes using model #1 are differentiated

Table 5.1: Performance results for various stylized rendering methods, across different scene sizes and camera positions. Scenes represented in this table all utilized curvature-based stroke rendering.

Curvature-Based Strokes		
Model #1 Wide Angle Shot	Avg Frame Time (ms)	Avg Frame Rate (FPS)
Phong Shading (Baseline)	7.92	126.26
BW Style	14.65	68.26
GN Style	12.81	78.06
OR Style	14.65	68.26
Model #1 Low Angle Shot	Avg Frame Time (ms)	Avg Frame Rate (FPS)
Phong Shading (Baseline)	5.73	174.52
BW Style	8.67	115.34
GN Style	7.44	134.41
OR Style	8.71	114.81
Model #2 Mountain Shot	Avg Frame Time (ms)	Avg Frame Rate (FPS)
Phong Shading (Baseline)	16.53	60.50
BW Style	20.42	48.97
GN Style	23.34	42.84
OR Style	20.57	48.61

Table 5.2: Performance results for various stylized rendering methods, across different scene sizes and camera positions. Scenes represented in this table all utilized hybrid-overlay stroke rendering.

Hybrid-Overlay Strokes		
Model #1 Wide Angle Shot	Avg Frame Time (ms)	Avg Frame Rate (FPS)
Phong Shading (Baseline)	7.92	126.26
BW Style	25.72	38.88
GN Style	23.65	42.28
OR Style	25.37	39.42
Model #1 Low Angle Shot	Avg Frame Time (ms)	Avg Frame Rate (FPS)
Phong Shading (Baseline)	5.73	174.52
BW Style	16.71	59.84
GN Style	15.51	64.47
OR Style	16.53	60.49
Model #2 Mountain Shot	Avg Frame Time (ms)	Avg Frame Rate (FPS)
Phong Shading (Baseline)	16.53	60.50
BW Style	36.97	27.05
GN Style	41.25	24.24
OR Style	37.97	26.34

by the breadth of the camera shot, and consequently the amount of visible terrain in terms of surface area. The lowest observed frame rate for Model #1 was around 40FPS using the chaining-based stroke rendering, which lines up with the "wide-angle shot" trial in Figure 8. Model #2 was largely used as a stress-test, and saw significantly lower frame rates overall; however the difference between our implementation and the baseline rendering method was comparable.

The impact of the different stroke-rendering styles varies depending on the scene, but as expected the hybrid-overlay (or chaining) method performs worse overall. For model #1, the curvature-based method obtained around 30 more frames per second than the chaining method, equivalent to around 10ms of additional frame time. Similarly, the larger model #2 saved around 15ms of frame time by using the curvature-based method, likely due to the increased number of edges present in the scene. These results are indicative of the increased complexity of the chaining algorithm used by the hybrid-overlay method, though both methods still demonstrate interactive frame rates when rendering modest amounts of geometry.

5.2 Visual Results

Visually, the topic of stylized rendering is inherently subjective, as it deals with human-borne concepts including style and visual appeal. As a result, proper validation of visual output is tricky and often done purely through demonstration of output frames. However, for this project we also employ a short user study to provide basic confirmation of our visual results and gauge comparability to hand-drawn art through a human lens. The following screenshots were rendered using various style parameters at a resolution of approximately 1920x1080. Figures 5.2 and 5.3 show a few different

Figure 5.2: A cartoonish style with a custom skybox.

styles that utilize hatching and curvature-based stroke rendering. Figures 5.4, 5.5, 5.6, and 5.7 demonstrate other style options that utilize hybrid-overlay strokes.

5.3 User Study

While evaluation of stylized visuals is inherently subjective, we performed a short user study to gather general feedback on our results, and allow us to compare a few different style options. We collected around 30 responses across two versions of a feedback form. Both versions contained the same 4 styles presented via short video clips, each with the same follow-up questions. The only difference between the two versions was that one used curvature-based strokes, and the other used hybrid-overlay strokes to render contours in the scene. These versions were distributed to different user groups, so that we could independently evaluate differences between the two stroke rendering technologies. Figures 5.8, 5.9, and 5.10 show results from our study regarding general visual appeal for various styles, in which users were expected to

Figure 5.3: A black and white sketch-like style with a closeup on some trees.

Figure 5.4: A black and white sketch-like style with a wide-angle shot. This version showcases hybrid-overlay strokes with a textures stroke mesh.

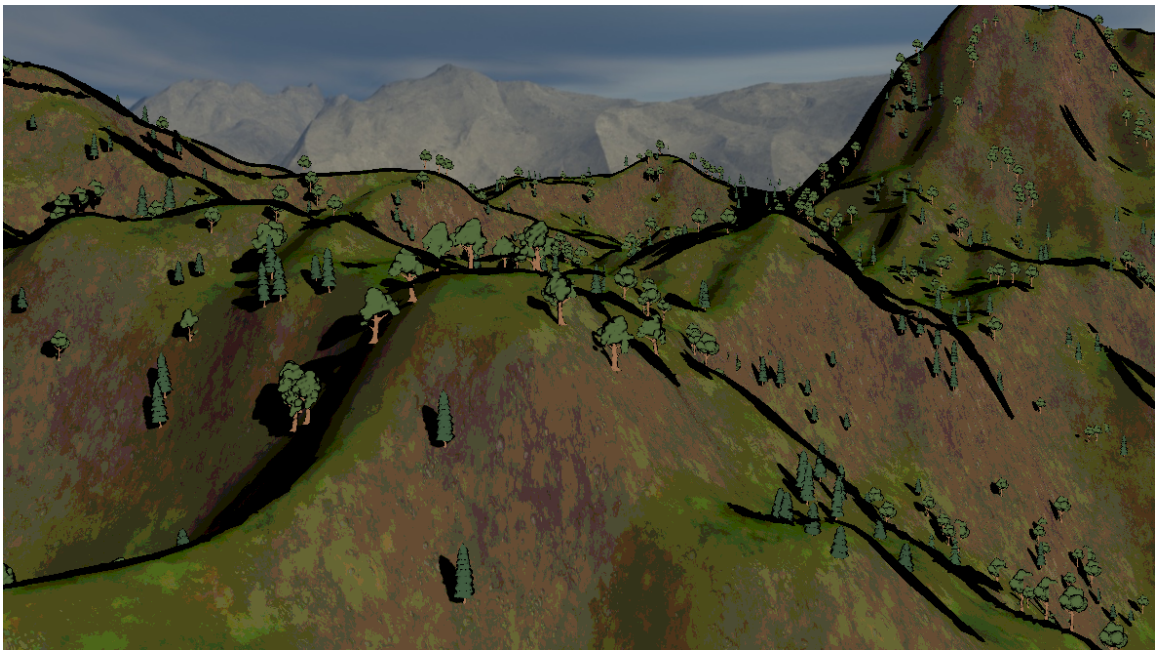


Figure 5.5: A style using terrain textures with modified contrast channels.

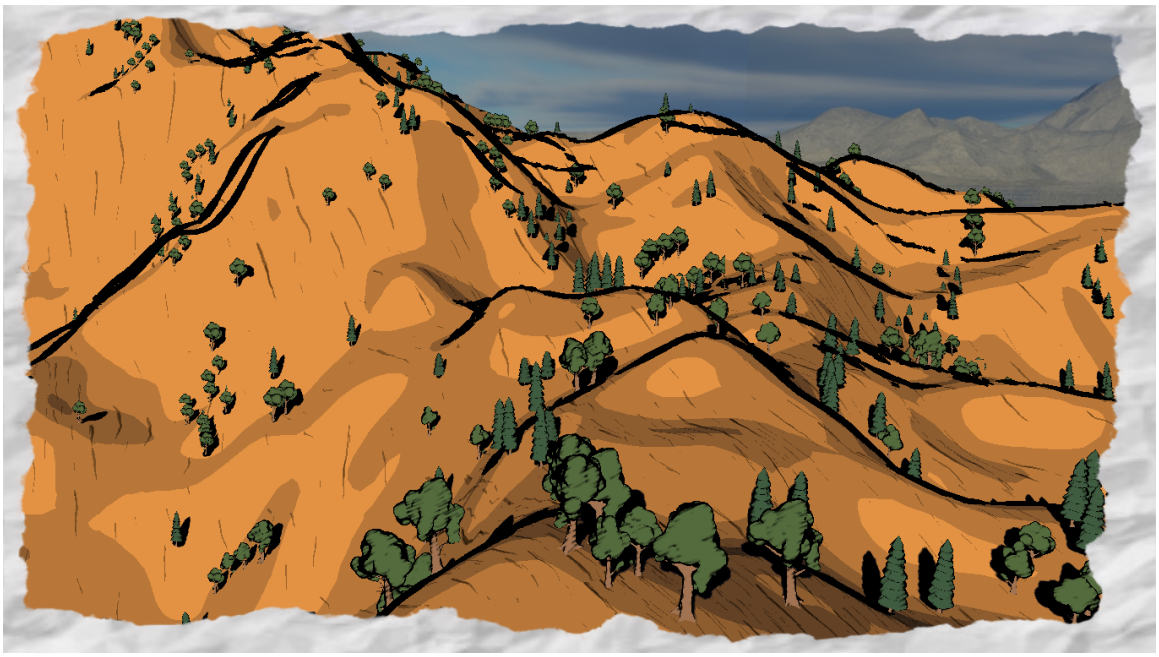


Figure 5.6: An orange style using heavy cell-shading, overlay strokes, and an overlay border.

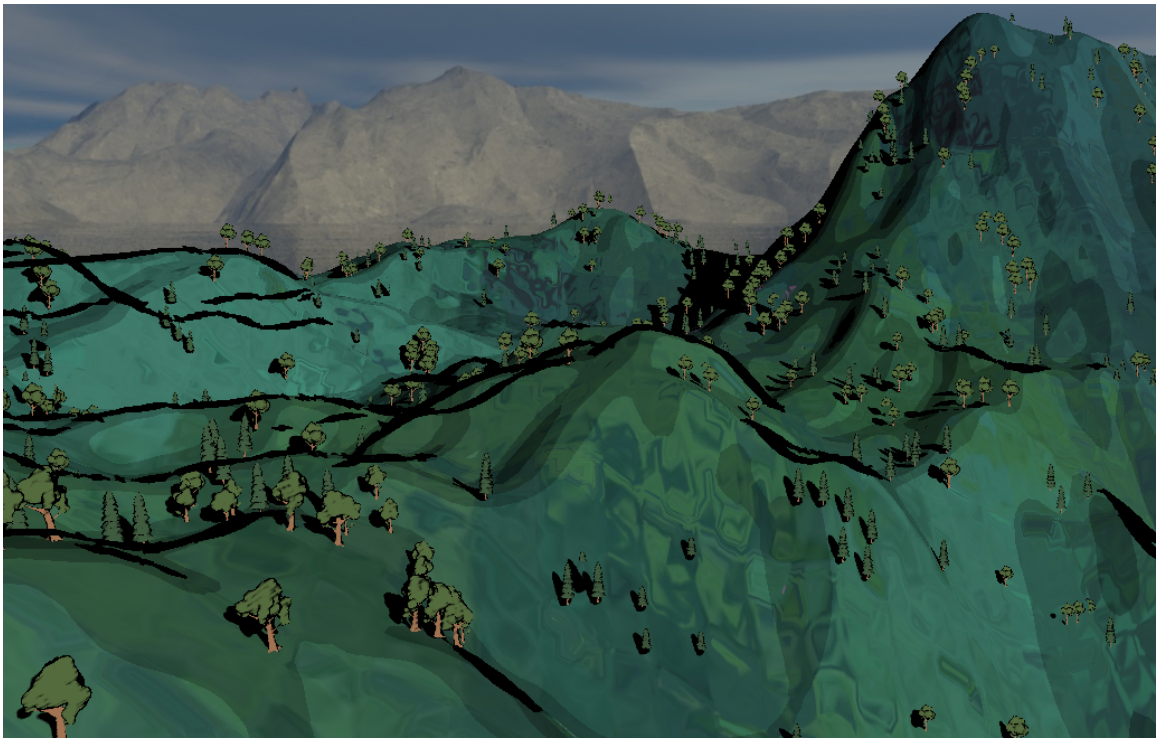
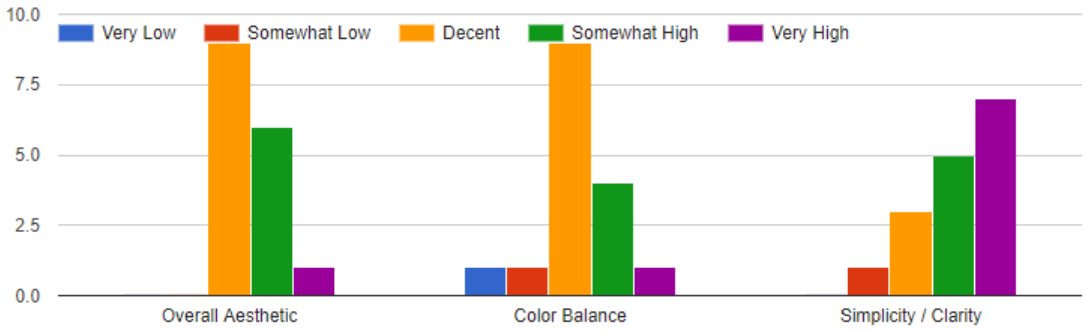


Figure 5.7: A watercolor-like style created by warping texture coordinates.

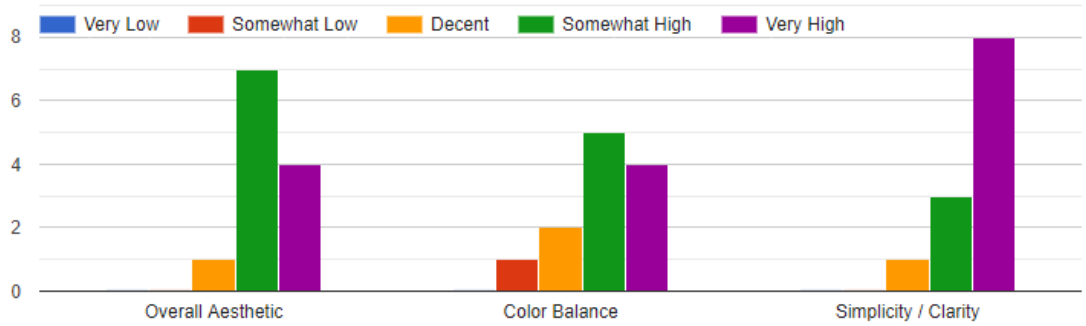
rate each scene based on overall aesthetic, color balance, and clarity. Figures 5.11, 5.12, and 5.13 show results regarding how well users thought the style achieved a hand-drawn or 2D look.

Overall, users who participated in our study mostly enjoyed the styles presented and thought that each style had at least a decent degree of comparability to hand-drawn art styles. This is evidenced by the fact that user responses for hand-drawn appearance was generally higher for the stylized scenes, than it was for the baseline Phong shaded scene. Additionally, our user study revealed some interesting patterns between the different styles and rendering technologies. Across the board, the styles with hybrid-overlay strokes scored lower than the styles with curvature-based strokes, both in terms of hand-drawn aesthetic and general appeal. This was partially expected due to the fact that hybrid-overlay strokes are visually noisy and don't have the same level of temporal coherence as curvature-based strokes. The only significant

(a) An example frame of the black-and-white style video shown to users. .



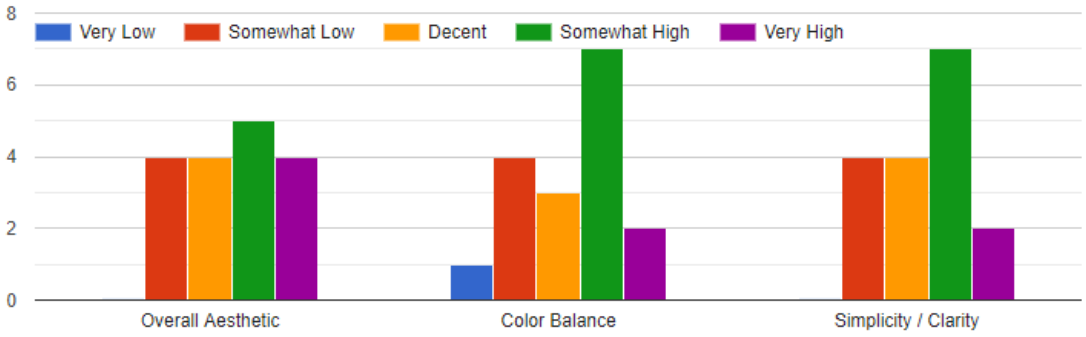
(b) User results using the hybrid-overlay stroke rendering method.



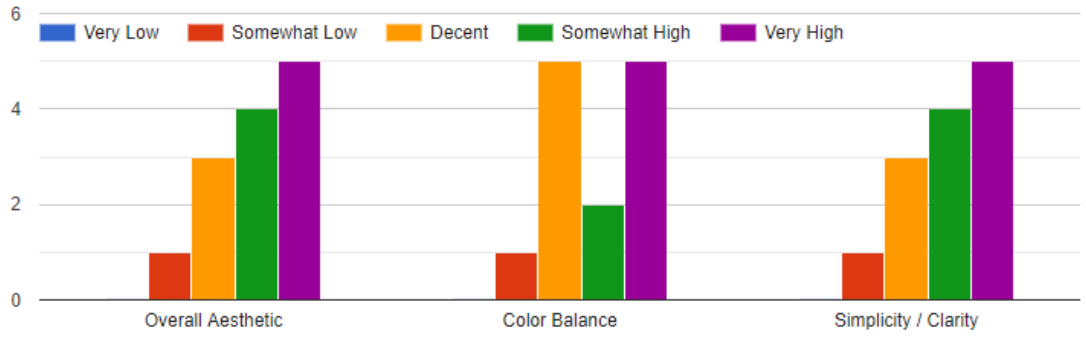
(c) User results using the curvature-based stroke rendering method.

Figure 5.8: User study results regarding general aesthetic of the black-and-white style. Users were prompted with: "Please evaluate the scene demonstrated in this video on the following qualities"

(a) An example frame of the green-terrain style video shown to users. .

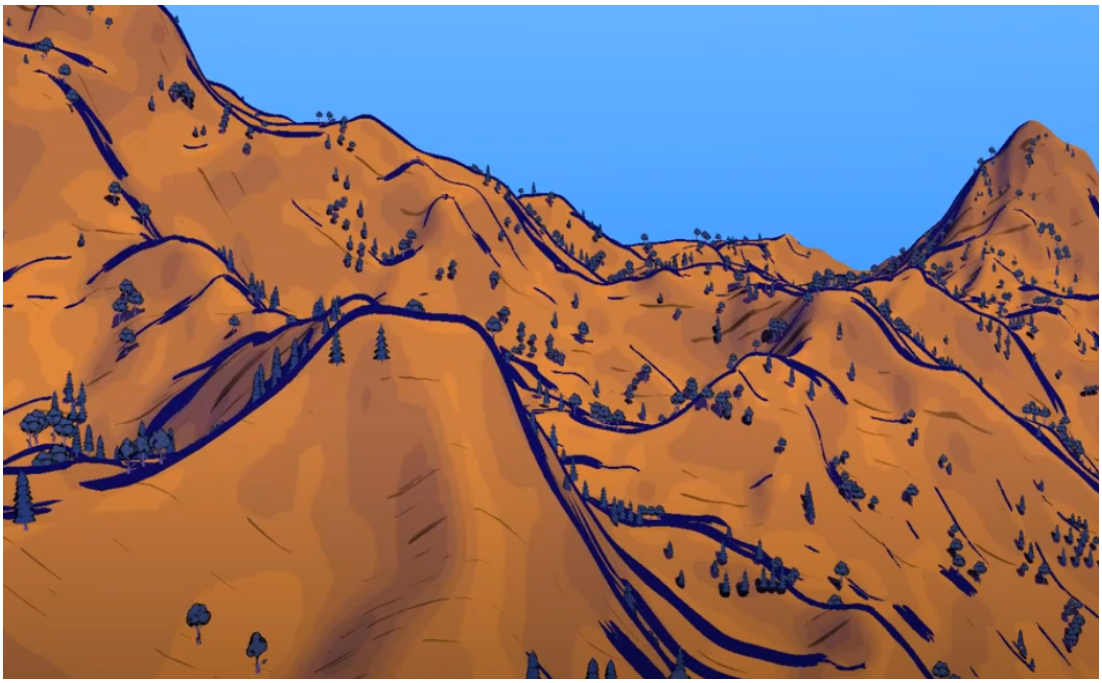


(b) User results using the hybrid-overlay stroke rendering method.

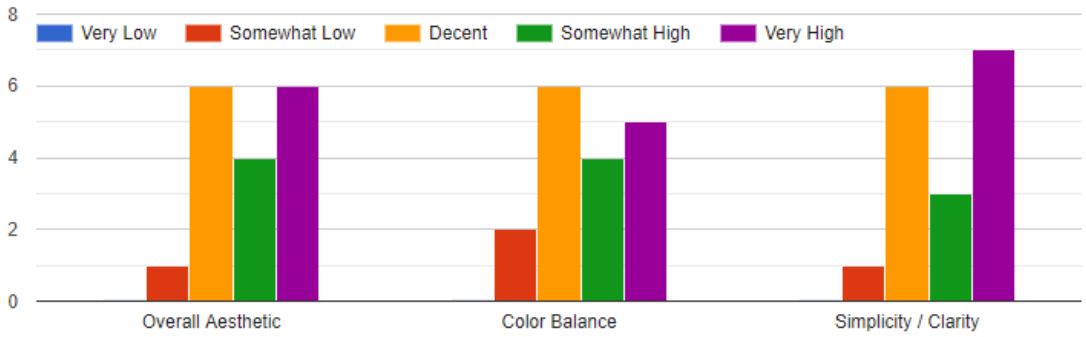


(c) User results using the curvature-based stroke rendering method.

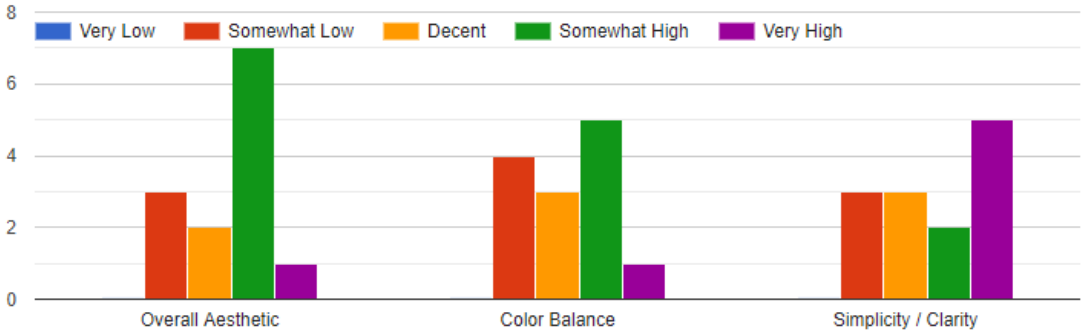
Figure 5.9: User study results regarding general aesthetic of the green-terrain style. Users were prompted with: "Please evaluate the scene demonstrated in this video on the following qualities"



(a) An example frame of the orange style video shown to users. .

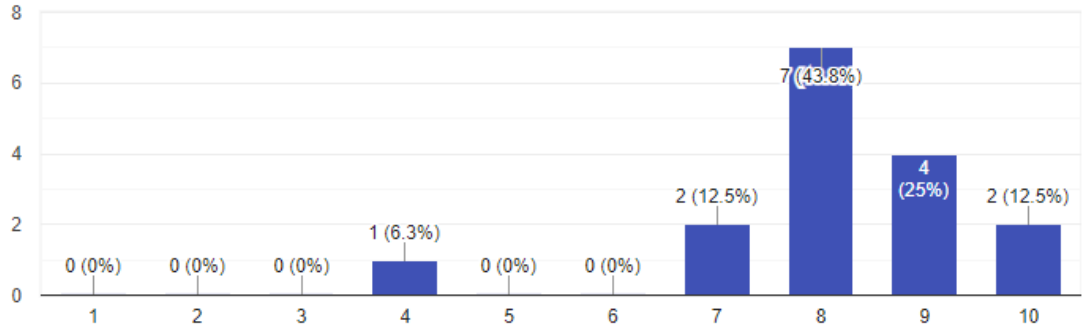


(b) User results using the hybrid-overlay stroke rendering method.

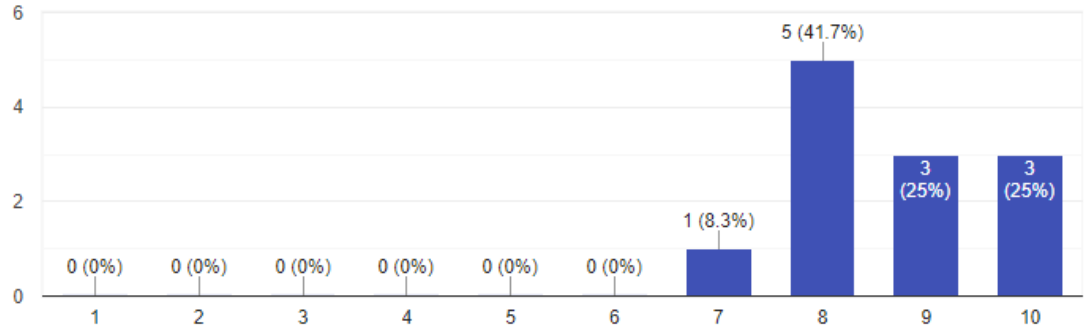


(c) User results using the curvature-based stroke rendering method.

Figure 5.10: User study results regarding general aesthetic of the orange style. Users were prompted with: "Please evaluate the scene demonstrated in this video on the following qualities"

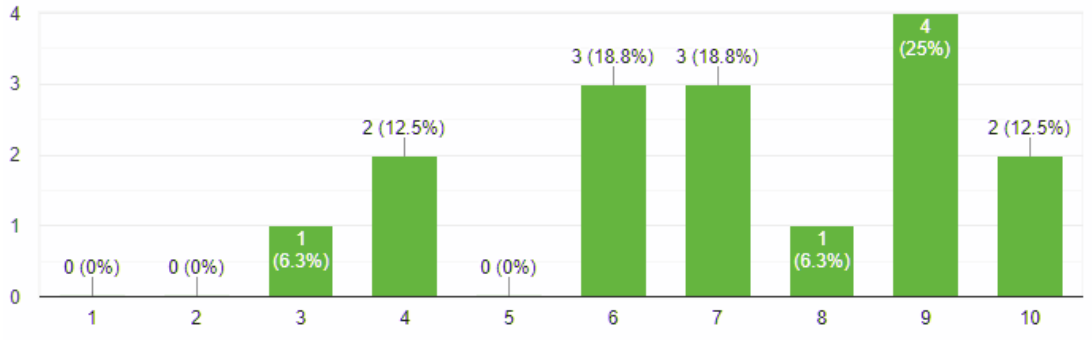


(a) User results using the hybrid-overlay stroke rendering method.

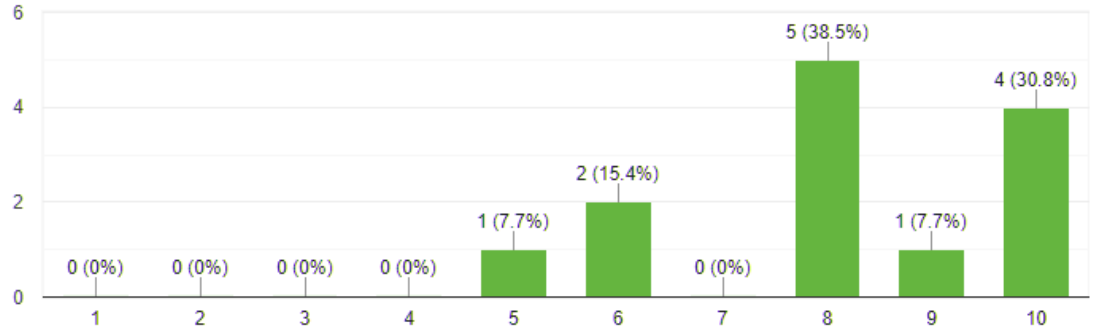


(b) User results using the curvature-based stroke rendering method.

Figure 5.11: User study results regarding how well the black-and-white style achieved a "hand-drawn" look. Users were prompted with: "How well do you think this scene simulates a hand-drawn or 2D art style?"

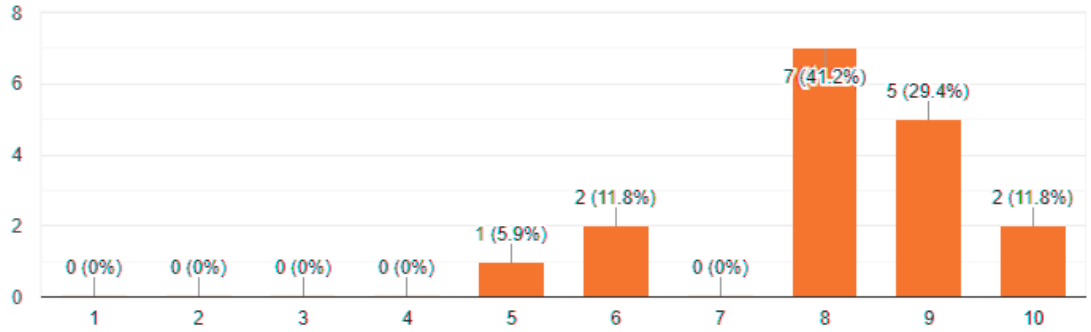


(a) User results using the hybrid-overlay stroke rendering method.

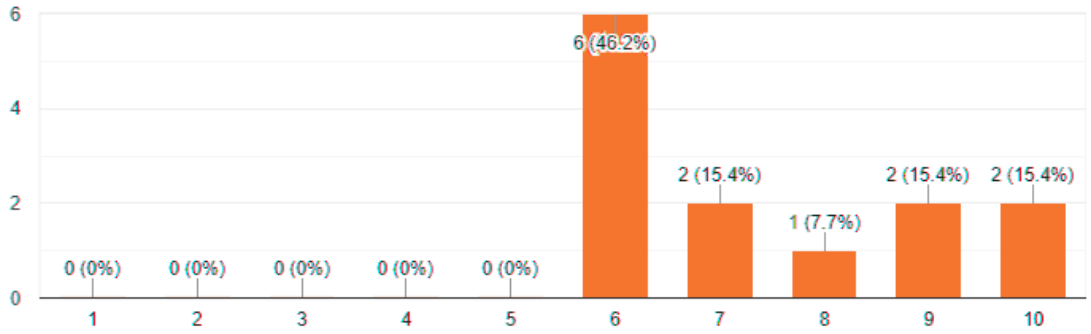


(b) User results using the curvature-based stroke rendering method.

Figure 5.12: User study results regarding how well the green-terrain style achieved a "hand-drawn" look. Users were prompted with: "How well do you think this scene simulates a hand-drawn or 2D art style?"



(a) User results using the hybrid-overlay stroke rendering method.



(b) User results using the curvature-based stroke rendering method.

Figure 5.13: User study results regarding how well the orange style achieved a "hand-drawn" look. Users were prompted with: "How well do you think this scene simulates a hand-drawn or 2D art style?"

exception to this pattern was for the orange style, where hybrid-overlay stroke rendering scored better in relation to its hand-drawn appearance. It is difficult to say whether this pattern is influenced by color palette or other aspects intrinsic to the orange-style scene; however, it's worth noting that some styles may lend themselves better to different stroke rendering methods.

Chapter 6

FUTURE WORK

Much of the implementation described in this paper is focused on obtaining unique stylized visuals while maintaining real-time performance. Modern CPUs and GPUs are improving significantly in terms of manageable workload and real-time performance, and this may open future options for more complex rendering algorithms to be used in this field. This includes using advanced curve-fitting algorithms to perform edge chaining, which currently costs too much in the way of computational resources. Similarly, it may be interesting to explore rendering methods that employ machine learning, including style-transfer networks which map certain art styles to arbitrary frames.

6.1 Hatching Limitations

The dynamic solid method for hatching used in this paper is limited in the kinds of textures that can be used. Often times textures with distinct curved lines or images become blurry when accessed from a dynamic solid, and they can sometimes lose important visual features as a result. Additionally, hatch textures can only be applied in a few global directions, due to the requirement of dynamic solids that inputs be continuous in world-space. Our implementation is able to represent a selection of hatch textures that add hand-drawn detail to the scene, but it would be interesting to see how improvements to the dynamic solid method might enable more flexibility. Some possible extensions to this technology include integration with

patch-based surface texturing, advanced surface interpolation methods, and screen-space alternatives.

Additionally, our implementation doesn't focus much on the creation of hatch-line textures, merely their placement in the scene. It might be beneficial to provide users with an easier way to generate novel hatch-line textures, including automatic generation of various shades or tones. In Section 4 we describe one method for overlaying hatch textures on top of each other to obtain darker shades, however a more complex blending method might provide coherence for a wider range of texture styles, such as stipple textures or high-detail images.

6.2 Coherence Limitations

The most noticeable limitation of this work is the visual noise produced by the overlay stroke rendering algorithm. As discussed in Section 4.3, temporal coherence is difficult to achieve in a setting where features change significantly between frames. We implement a few methods for masking noise, but a more comprehensive implementation of optical flow would likely help significantly. Some past papers, such as [24], introduce a more complex algorithm for temporally coherent stroke rendering; however, some considerations need to be made for integration with large-scale environments and performance limitations, as is the focus of this paper. In general, stylized rendering methods that have better temporal coherence often sacrifice customizability or performance, often due to the necessary shift towards sequential screen-space algorithms. We hope to see future works explore this area in more detail with respect to real-time rendering.

Another avenue for future exploration could be using intermediate representations for stroke-lines, such as parameterized curves, to more accurately track them in world

space. This idea is used in [15] to allow users to interactively place stroke lines on the surface of an object. However, this may be extended to automatic stroke line placement. Representing strokes as curves in world-space may allow the renderer to treat them as normal objects, and present them coherently over time. The challenge of determining when and how to update stroke lines will still remain, but perhaps it would be an easier task with intermediate stroke representations. Time constraints prevented us from exploring this topic in-depth, but we believe it could be a worthwhile endeavor for future research.

Chapter 7

CONCLUSION

In this paper we demonstrate an implementation of stylized rendering for large-scale environments, using various stroke rendering and shading techniques. Our implementation combines multiple existing technologies into a unified system for real-time rendering, including two different stroke rendering algorithms, hatch-line texturing with dynamic solids, and custom shading options. By using compute shaders and an improved chaining algorithm, we demonstrate improved performance over existing methods for stroke rendering, achieving interactive frame rates ranging from 40-70 FPS for average-sized scenes. In addition, we analyzed the visuals produced by our algorithm using a short user study, which highlighted some interesting patterns and differences between various styles and rendering technologies. Our hybrid-overlay strokes produced more distracting visuals than curvature-based strokes, likely due to a lack of temporal coherence, though they provide more flexibility in terms of creativity stylization options. Overall, we believe our system acts as an example of how stylized rendering systems can combine multiple complex technologies for feature extraction and representation, all while maintaining real-time frame rates for interactive entertainment applications. Art and style in computer graphics is an evolving field, and we hope to see such works expanded as related industries move forward.

BIBLIOGRAPHY

[1] glfw/glfw. https://github.com/glfw/glfw, June 2021. original-date: 2013-04-18T15:24:53Z.

[2] P. Barla, J. Thollot, and L. Markosian. X-toon: an extended toon shader. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, NPAR '06, pages 127–132, New York, NY, USA, June 2006. Association for Computing Machinery.

[3] L. Bavoil and K. Myers. Order independent transparency with dual depth peeling. Technical report, 2008.

[4] S. Bhattacharjee and P. Narayanan. Real-Time Painterly Rendering of Terrains. In *2008 Sixth Indian Conference on Computer Vision, Graphics Image Processing*, pages 568–575, Dec. 2008.

[5] A. Bousseau, M. Kaplan, J. Thollot, and F. X. Sillion. Interactive watercolor rendering with temporal coherence and abstraction. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, NPAR '06, pages 141–149, New York, NY, USA, June 2006. Association for Computing Machinery.

[6] P. Bénard, A. Bousseau, and J. Thollot. THOLLOT J.: Dynamic solid textures for real-time coherent stylization. In *In Proceedings of the 2009 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*, pages 121–127. ACM press.

[7] P. Bénard, A. Bousseau, and J. Thollot. Dynamic solid textures for real-time coherent stylization. In *Proceedings of the 2009 symposium on Interactive*

3D graphics and games - I3D '09, page 121, Boston, Massachusetts, 2009. ACM Press.

[8] O. Cornut. Dear Imgui. https://github.com/ocornut/imgui, June 2021. original-date: 2014-07-21T14:29:47Z.

[9] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. Suggestive Contours for Conveying Shape. page 8, 2003.

[10] K. Engel, M. Hadwiger, J. M. Kniss, A. E. Lefohn, C. R. Salama, and D. Weiskopf. Real-time volume graphics. In *ACM SIGGRAPH 2004 Course Notes*, SIGGRAPH '04, pages 29–es, New York, NY, USA, Aug. 2004. Association for Computing Machinery.

[11] A. Hertzmann. Painterly rendering with curved brush strokes of multiple sizes. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques - SIGGRAPH '98*, pages 453–460, Not Known, 1998. ACM Press.

[12] A. Hertzmann. A survey of stroke-based rendering. *IEEE Computer Graphics and Applications*, 23(4):70–81, July 2003.

[13] T. Judd, F. Durand, and E. Adelson. Apparent ridges for line drawing. *ACM Transactions on Graphics*, 26(3):19–es, July 2007.

[14] R. D. Kalnins, P. L. Davidson, L. Markosian, and A. Finkelstein. Coherent Stylized Silhouettes. page 6.

[15] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein. WYSIWYG NPR: Drawing Strokes Directly on 3D Models. page 8.

[16] H. Kang, S. Lee, and C. K. Chui. Coherent line drawing. In *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*, NPAR '07, pages 43–50, New York, NY, USA, Aug. 2007. Association for Computing Machinery.

[17] H. Kang, S. Lee, and C. K. Chui. Flow-Based Image Abstraction. *IEEE Transactions on Visualization and Computer Graphics*, 15(1):62–76, Jan. 2009. Conference Name: IEEE Transactions on Visualization and Computer Graphics.

[18] H. W. Kang, C. K. Chui, and U. K. Chakraborty. A unified scheme for adaptive stroke-based rendering. *The Visual Computer*, 22(9-11):814–824, Sept. 2006.

[19] A. W. Klein, W. Li, M. M. Kazhdan, W. T. Corrêa, A. Finkelstein, and T. A. Funkhouser. Non-photorealistic virtual environments. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques - SIGGRAPH '00*, pages 527–534, Not Known, 2000. ACM Press.

[20] M. A. Kowalski, L. Markosian, J. D. Northrup, L. Bourdev, R. Barzel, L. S. Holden, and J. F. Hughes. Art-based rendering of fur, grass, and trees. In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques - SIGGRAPH '99*, pages 433–438, Not Known, 1999. ACM Press.

[21] A. Lake, C. Marshall, M. Harris, and M. Blackstein. Stylized rendering techniques for scalable real-time 3D animation. In *Proceedings of the first international symposium on Non-photorealistic animation and rendering - NPAR '00*, pages 13–20, Annecy, France, 2000. ACM Press.

[22] K. Lawonn, I. Viola, B. Preim, and T. Isenberg. A Survey of Surface-Based Illustrative Rendering for Visualization: Surface-Based Illustrative Rendering. *Computer Graphics Forum*, 37(6):205–234, Sept. 2018.

[23] H. Lee, S. Kwon, and S. Lee. Real-time pencil rendering. In *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*, NPAR '06, pages 37–45, New York, NY, USA, June 2006. Association for Computing Machinery.

[24] L. Lou, L. Wang, and X. Meng. Stylized strokes for coherent line drawings. *Computational Visual Media*, 1(1):79–89, Mar. 2015.

[25] L. Markosian. *Art-based modeling and rendering for computer graphics*. phd, Brown University, USA, 2000. AAI9987803 ISBN-10: 0599941634.

[26] B. J. Meier. Painterly rendering for animation. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques - SIGGRAPH '96*, pages 477–484, Not Known, 1996. ACM Press.

[27] J. D. Northrup and L. Markosian. Artistic silhouettes: a hybrid approach. In *Proceedings of the first international symposium on Non-photorealistic animation and rendering - NPAR '00*, pages 31–37, Annecy, France, 2000. ACM Press.

[28] Y. Ohtake, A. Belyaev, and H.-P. Seidel. Ridge-valley lines on meshes via implicit surface fitting. In *ACM SIGGRAPH 2004 Papers*, SIGGRAPH '04, pages 609–612, New York, NY, USA, Aug. 2004. Association for Computing Machinery.

[29] 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, page 581, Not Known, 2001. ACM Press.

[30] S. Rusinkiewicz. trimesh2. https://gfx.cs.princeton.edu/proj/trimesh2/.

[31] P.-p. J. Sloan, W. Martin, A. Gooch, and B. Gooch. *The Lit Sphere: A Model for Capturing NPR Shading from Art*. 2001.

[32] M. Webb, E. Praun, A. Finkelstein, and H. Hoppe. Fine tone control in hardware hatching. page 7.

[33] B. Whited, E. Daniels, M. Kaschalk, P. Osborne, and K. Odermatt. Computer-assisted animation of line and paint in Disney's Paperman. Aug. 2012.

[34] H. Xu and B. Chen. Stylized rendering of 3D scanned real world environments. In *Proceedings of the 3rd international symposium on Non-photorealistic animation and rendering - NPAR '04*, page 25, Annecy, France, 2004. ACM Press.

APPENDICES

Appendix A

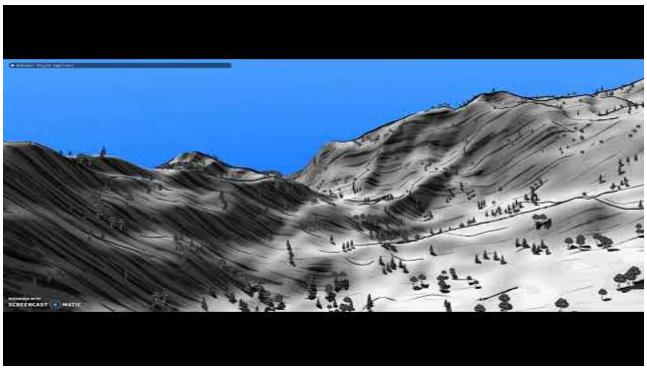
VISUAL RESULTS SURVEY

Stylized Rendering Feedback Form

The intention of this survey is to evaluate the visual appeal of various stylization techniques in a 3D scene.

Please watch the video below and respond to the questions that follow. Pay attention to aspects of the scene which catch your eye, or look interesting/distracting. Feel free to zoom in if it's difficult to see the video (sometimes full screen does not work). Thanks for your time!

Style #1 -- Black & White (https://youtu.be/dn_ziibSqro)



http://youtube.com/watch?v=dn_ziibSqro

1. Please evaluate the scene demonstrated in this video on the following qualities

Mark only one oval per row.

	Very Low	Somewhat Low	Decent	Somewhat High	Very High
Overall Aesthetic	◯	◯	◯	◯	◯
Color Balance	◯	◯	◯	◯	◯
Simplicity / Clarity	◯	◯	◯	◯	◯

2. How well do you think this scene simulates a hand-drawn or 2D art style?

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very Poorly	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very Well

3. What aspects of the style contributed or detracted from a hand-drawn or 2D appearance?

4. Did you find the style distracting or disorienting?

Mark only one oval.

- ◯ No, it was smooth & easy to look at
- ◯ Yes, it was somewhat disorienting
- ◯ Yes, it was very disorienting

5. What is/are your favorite aspect(s) of the scene? (optional)

Stylized Rendering Feedback Form - 2 of 4

Please watch the video below and respond to the questions that follow. Pay attention to aspects of the scene which catch your eye, or look interesting/distracting. Feel free to zoom in if it's difficult to see the video. Thanks for your time!

Style #2 -- Green Terrain (https://youtu.be/u15szLA0lg0)

http://youtube.com/watch?v=u15szLA0lg0

6. Please evaluate the scene demonstrated in this video on the following qualities

Mark only one oval per row.

	Very Low	Somewhat Low	Decent	Somewhat High	Very High
Overall Aesthetic	◯	◯	◯	◯	◯
Color Balance	◯	◯	◯	◯	◯
Simplicity / Clarity	◯	◯	◯	◯	◯

7. How well do you think this scene simulates a hand-drawn or 2D art-style?

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very Poorly	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very Well

8. What aspects of the style contributed or detracted from a hand-drawn or 2D appearance?

9. Did you find the style distracting or disorienting?

Mark only one oval.

- No, it was smooth & easy to look at
- Yes, it was somewhat disorienting
- Yes, it was very disorienting

10. What is/are your favorite aspect(s) of the scene? (optional)

Stylized Rendering Feedback Form - 3 of 4

Please watch the video below and respond to the questions that follow. Pay attention to aspects of the scene which catch your eye, or look interesting/distracting. Feel free to zoom in if it's difficult to see the video. Thanks for your time!

Style #3 -- Simple Shading (https://youtu.be/Dat5rE9EygY)

http://youtube.com/watch?v=Dat5rE9EygY

11. Please evaluate the scene demonstrated in this video on the following qualities

Mark only one oval per row.

	Very Low	Somewhat Low	Decent	Somewhat High	Very High
Overall Aesthetic	◯	◯	◯	◯	◯
Color Balance	◯	◯	◯	◯	◯
Simplicity / Clarity	◯	◯	◯	◯	◯

12. How well do you think this scene simulates a hand-drawn or 2D art style?

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very Poorly	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very Well

13. What aspects of the style contributed or detracted from a hand-drawn or 2D appearance?

14. Did you find the style distracting or disorienting?

Mark only one oval.

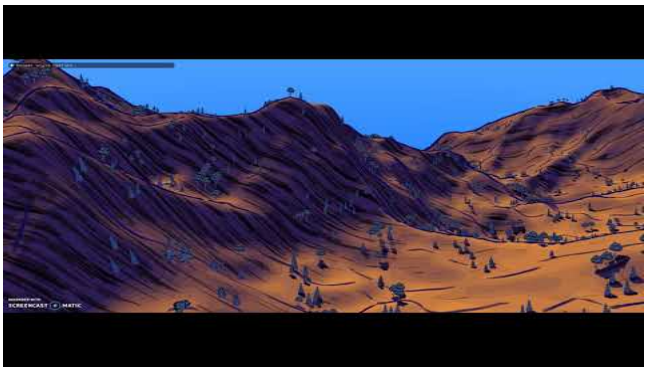
- No, it was smooth & easy to look at
- Yes, it was somewhat disorienting
- Yes, it was very disorienting

15. What is/are your favorite aspect(s) of the scene? (optional)

Stylized Rendering Feedback Form - 4 of 4

Please watch the video below and respond to the questions that follow. Pay attention to aspects of the scene which catch your eye, or look interesting/distracting. Feel free to zoom in if it's difficult to see the video. Thanks for your time!

Style #4 -- Orange/Blue Shift (https://youtu.be/WA8VdPSM3iA)



http://youtube.com/watch?v=WA8VdPSM3iA

16. Please evaluate the scene demonstrated in this video on the following qualities

Mark only one oval per row.

	Very Low	Somewhat Low	Decent	Somewhat High	Very High
Overall Aesthetic	◯	◯	◯	◯	◯
Color Balance	◯	◯	◯	◯	◯
Simplicity / Clarity	◯	◯	◯	◯	◯

17. How well do you think this scene simulates a hand-drawn or 2D art-style?

Mark only one oval.

	1	2	3	4	5	6	7	8	9	10	
Very Poorly	◯	◯	◯	◯	◯	◯	◯	◯	◯	◯	Very Well

18. What aspects of the style contributed or detracted from a hand-drawn or 2D appearance?

19. Did you find the style distracting or disorienting?

Mark only one oval.

- ◯ No, it was smooth & easy to look at
- ◯ Yes, it was somewhat disorienting
- ◯ Yes, it was very disorienting

20. What is/are your favorite aspect(s) of the scene? (optional)