Table of Contents

[1. Abstract 2](#_Toc118624118)

[2. Acknowledgements 2](#_Toc118624119)

[3. Introduction 2](#_Toc118624120)

[4. Graphical Processing Units 2](#_Toc118624121)

[4.1. Introduction 2](#_Toc118624122)

[4.2. Pipeline 2](#_Toc118624123)

[4.3. Writing Shaders 2](#_Toc118624124)

[5. Vector Graphics 2](#_Toc118624125)

[5.1. Introduction 2](#_Toc118624126)

[5.2. Spline, B-Spline and Bezier Curve 2](#_Toc118624127)

[6. GPU rendering of Splines 3](#_Toc118624128)

[6.1. Loop and Blinn 3](#_Toc118624129)

[6.2. Pre tessellation 4](#_Toc118624130)

[7. Technology 4](#_Toc118624131)

[7.1. Introduction 4](#_Toc118624132)

[7.2. Shader Creation tools 4](#_Toc118624133)

[7.2.1. Unreal Engines 4](#_Toc118624134)

[7.2.2. Unity 4](#_Toc118624135)

[7.2.3. Godot 5](#_Toc118624136)

[7.3. GPU integration & profiling 5](#_Toc118624137)

[7.4. Spline tools 5](#_Toc118624138)

[7.4.1. Unreal Engine 5](#_Toc118624139)

[7.4.2. Unity 5](#_Toc118624140)

[7.4.3. Godot 5](#_Toc118624141)

[8. Methodology & Design 6](#_Toc118624142)

[9. Implementation 6](#_Toc118624143)

[9.1. Sprints 6](#_Toc118624144)

[10. Results & Conclusions 6](#_Toc118624145)

[11. References 6](#_Toc118624146)

# Abstract

# Acknowledgements

* Peter Given
* Claire Horgan
* Robert Sheehy

# Introduction

Tools OLM Digital R&D’s Shader painter’s method (Petikam, Anjyo and Rhee, 2021) for manually adding shadows to characters in real time digital worlds enabled for enabled for more dynamic non photo realistic shading in games, interactive media, and digital movies. The method had limited is range of shapes for shadows. A function is used to define these shapes. Rectangular are impossible to create using the function. “…for mechanical objects such as robots, weapons, and cars, our parameters may have limited support for producing more complex edits for these geometric surfaces.”(Petikam, Anjyo and Rhee, 2021)

Using vector graphic to describe the shadow shapes would allow to create rectangles, the shapes available via the function and others. The reason for not using these was that “[vector shadows] offers limited dynamic shape manipulation without costly re-computation of the mesh.” This paper will attempt to find methods to avoid this limitation.

# Graphical Processing Units

## Introduction

## Pipeline

## Writing Shaders

# Vector Graphics

## Introduction

Vector graphics are a form of digital images. This format specifies the boundaries of shapes within the image as opposed to individual pixels as in the bitmap format. Vector images must be converted to pixels at runtime. This is called rendering. Rendering at runtime means the rendering process can be altered to the given range of the task being done. This way vectors can be rendered at any resolution while retaining clarity and sharpness. This disconnect between resolution and the data required means that high resolution images can be created from files with small memory sizes.

## Spline, B-Spline and Bezier Curve

In vector images, shape boundaries are made up of lines and splines. Splines are any curved line. Splines come in one of 2 varieties: open and closed. Closed splines enclose an area completely by having their 1st and last point overlap resulting in a continuous boarder with no breaks. Open splines have a distinct 1st and last points with non-identical positions. B-splines are a component of the vector shapes in the SVG format. (SVG Group, 2022)These splines are defined by a series of Bezier curves.

Bezier method of defining curves by a method of interpolating between a set of points and then interpolating the result. For a quadratic curve, 3 points are be used. The points are referred to as control points. For the given series A, B, and C; A would be interpolated with B to get Z and B would be interpolated with C to get Y. Then Z and Y would be interpolated to get the point on the curve. Each of these interpolations would be done with the same value of T. T is increases as you move along the curve. T of 0 is the beginning of the curve, T of .5 is halfway through the curve and T of 1 is the end of the curve. Cubic Bezier curves have 4 control points. Z is then the result of interpolating A and B and Y is the result of C and D.

Two graphs: a and b. 
Graph a shows points A,B, and C. Points Z and Y lay on the lines AB and BC. A final point lays on the line ZY.

Graph b shows points A,B,C, and D. Points Z and Y lay on lines AB and CD respectively. The final point is on line ZY.

Figure (a) Quadratic Bezier curve. (b) Cubic Bezier curve.

Cubic curves are used more commonly due to their finer control over the final curve. The function using to sample these curves is cubic, so it is more computationally intensive then the quadratic function of the quadratic Bezier curve.

# GPU rendering of Splines

## Loop and Blinn

Many methods for rendering vector graphics work of off the Loop and Blinn method. (Loop and Blinn, 2005) For this method, the GPU’s polygon processing capabilities are adapted to handle vectors. The control points of curves are passed into the GPU as vertices of a polygon. This vector mimicking a polygon is then processed as usual through the shader stages until the fragment shaders. The fragment shader varies depending on what type of curve was given.

Function – Quadratic Bezier curve inner bound equation (Loop and Blinn, 2005).

For a quadratic curve, the points on the quadratic curve would be given texture coordinates of (0,0) and (1,1) while the curve point would be given the point (0.5,0). The fragment shader will be provided an interpolated point between these 3 depending on the proximity of the fragment in texture space. This interpolated coordinate can be used in the function to determine whether the pixel is withing or outside the curve. This value can be used as the alpha, hiding either any fragment inside or any fragment outside the curve depending on its sign.

Efficiency is increased if overlapping shapes are removed. This can be done by subdividing a curve into two, decreasing the number of pixels that are rendered and then drawn over. This subdividing is recommended to be prepossessed on the CPU and saved.

Some sets of 3 vertices, referred to as a triangle, may be fully filled. They can either be rendered traditionally or be given the alternate texture coordination of (0,1),(0,1), and (0,1).

## Pre tessellation

A method (Kumar and Sud, 2019) was described that attempts to increase efficiency of the Loop and Blinn method (Loop and Blinn, 2005) by avoiding the complexities of rendering cubic curves. This method Uses the tessellation stage of the GPU pipeline to approximate the cubic curve into quadratic curves based on a predefined error tolerance, signified by K. This method removes the need for the branching required in the fragment shader to handle the different cases described in Loop and Blinn (Loop and Blinn, 2005). The quadratic curve means the fragment shader can use the simpler quadratic function (Equation 1). Some of the computation in this method, compared to Loop and Blinn (Loop and Blinn, 2005) is moved from the fragment shader to the tessellation shader. This means operations may be done on a per vertex basis as opposed to correlating to the number of pixels.

# Technology

## Introduction

Video game engines are frameworks that allow for easier creation of digital and interactive environments. These frameworks custom tools for interacting with their GPU pipeline. These tools can allow for quicker development time for software which use the GPU for real-time graphical purposes. These include tools for creating shaders, creating vectors in the engine and profiling GPU performance.

## Shader Creation tools

Each of the 3 frameworks explored in this paper have a node-based shader tool: Unreal Engine’s Material Editor, Unity’s Shader Graph and Godot’s VisualShaders. These systems use a visual drag-and-drop method to code the shaders. In these systems, functions and operators written verbally in code are replaced by nodes, sometimes referred to as blocks, in a two-dimensional space. Data is passed from one node to another by connecting points on two or more nodes via a line. These remove much of the boilerplate work required for implement shaders in engines. None of these allow for the creation of tessellation nor geometry shaders and so are not suitable for the use case of this paper.

High Level Shader Language (HLSL) and OpenGL Shader Language (GLSL) are text-based methods for describing shader code. These are more feature complete as they are standards for interacting with GPU middleware directly. Many engines require additional boilerplate work, such as extra lines of code or entire additional classes, to insert custom text shader code into the engine’s GPU pipeline.

### Unreal Engines

Much of Unreal Engine’s documentation focuses on the node-based Material Editor (Epic Games, 2022a). Unreal Engine has a HLSL based shader creation functionality. This method requires the creation of a C++ class inheriting from the FShader class. (Epic Games, 2022b) The only tutorial that was found for this paper on the creation of geometry shaders for the Unreal Engine is by YivanLee and is in Chinese. (YivanLee, 2022)

### Unity

Unity wraps HLSL shader code in its own shader language wrapper called ShaderLab. (Unity Technologies, 2022b) This custom language requires a name for the shader collection. This language requires boilerplate code such as defining data required from the engine both in the ShaderLab section and within the HLSL. ShaderLab allows for any shader type described in the DirectX standard to be used, including hull, tessellation, domain, and geometry shaders. (Geometry Shaders Made Easy In Unity URP! For Those New to Shaders! ✔️ 2020.3 | Game Dev Tutorial, 2020)

### Godot

Godot allows for GLSL to be written to shader files directly in the editor. These shader files can be attached to material files to be applied to be used. This implementation features no boilerplate work but is limited to a subset of the GLSL features. Godot does not possess tessellation nor geometry shaders. CPU based mesh editing is suggested as an alternative in Godot’s documentation. (Godot, 2022b)

## GPU integration & profiling

Both Unity and Unreal Engine have GPU profiling functionality in each engine’s profiling tools. These profiling tools can show the duration rendering each shader tool to enable comparison between shaders. (Epic Games, 2022c),(Unity Technologies, 2022a) Godot’s profiler does not provide data on individual shaders. Third party tools, such as Nvidia’s Nsight, can provide as much details as Unity and Unreal Engine’s. Nsight is limited to Nvidia Graphics cards. (Nvidia, 2018)

## Spline tools

Each engine allows for additions to build in more functionality into the engine. Tools are required to define the spline in the digital world. Such a tool must be able to visualise the spline and provide means of moving the control points that make up the spline. In software solutions, the GUI means of moving the control points are commonly called handles.

### Unreal Engine

Unreal Engine’s first party solution for splines is meant to deform meshes. It uses a very efficient linear spline system. (Spline Tool - Unreal Engine 4 Tutorial - Easy Way to Create Ropes and Chains, 2020) Linear splines are very limited in their curve control.

Upside Down does propose a method for both quadratic and cubic b-splines. (Spline Tool - Unreal Engine 4 Tutorial - Easy Way to Create Ropes and Chains, 2020) This method in not as well integrated into the engine as the first party method. The objects from the Upside Down’s method are not automatically optimised when building a release build. They use generic objects as handles so there is less visual feedback on the impact the control points have on the line. It does have in world visuals for the spline.

### Unity

Unity only has third party spline creation tools. The custom editor tool creation functionality in Unity is in depth allowing for third party solutions. Code Monkey described a spline solution. (Splines are Awesome!!!, 2020) This method has handles in the virtual space and visible curves. The implementation described is only cubic.

### Godot

Godot has first party splines, referred to as paths. (Godot, 2022a) This is only a cubic implementation. These paths have clean spline visuals and handles. Godot’s editor tools are scripts which can run in the editor without the game world running. These tools could be used to implement a custom spline solution. No existing third-party solutions for splines were found during research.

# Methodology & Design

# Implementation

## Function specification

|  |
| --- |
| Must have |
| Curve creation UI (quadratic) |
| Curve rendering shader (quadratic) |
| Curve rendering in Unity |
| Should have |
| Curve creation UI (cubic) |
| Curve rendering shader (cubic) |
| Curve rendering in Unity (cubic) |
| Curve defined shapes |
| Could have |
| Cell based curve rendering shader |
| Curve animating |
| Curve animating based on light direction |
| Have shape conform to geometry |
| Won’t have |
| Multiple blending systems |
| CMYK colour |
| Textured fills |
| Gradient fills |

# Results & Conclusions

# References

Epic Games, 2022a. *Materials*. [online] Available at: <https://docs.unrealengine.com/4.27/en-US/RenderingAndGraphics/Materials/> [Accessed 1 November 2022].

Epic Games, 2022b. *Shader Development*. [online] Available at: <https://docs.unrealengine.com/5.0/en-US/shader-development-in-unreal-engine/> [Accessed 1 November 2022].

Epic Games, 2022c. *Timing Insights*. [online] Available at: <https://docs.unrealengine.com/5.0/en-US/timing-insights-in-unreal-engine-5/> [Accessed 26 October 2022].

*Geometry Shaders Made Easy In Unity URP! For Those New to Shaders! ✔️ 2020.3 | Game Dev Tutorial*. 2020. Available at: <https://www.youtube.com/watch?v=7C-mA08mp8o> [Accessed 27 October 2022].

Godot, 2022a. *Beziers, curves and paths*. [online] Godot Engine documentation. Available at: <https://docs.godotengine.org/en/stable/tutorials/math/beziers\_and\_curves.html> [Accessed 26 October 2022].

Godot, 2022b. *Procedural geometry*. [online] Godot Engine documentation. Available at: <https://docs.godotengine.org/en/stable/tutorials/3d/procedural\_geometry/index.html> [Accessed 27 October 2022].

Kumar, H. and Sud, A., 2019. Fast, memory efficient and resolution independent rendering of cubic Bézier curves using tessellation shaders. In: *SIGGRAPH Asia 2019 Posters*, SA ’19. [online] New York, NY, USA: Association for Computing Machinery. pp.1–2. https://doi.org/10.1145/3355056.3364548.

Loop, C. and Blinn, J., 2005. Resolution independent curve rendering using programmable graphics hardware. *ACM Transactions on Graphics*, 24(3), pp.1000–1009. https://doi.org/10.1145/1073204.1073303.

Nvidia, 2018. *NVIDIA Nsight Systems*. [online] NVIDIA Developer. Available at: <https://developer.nvidia.com/nsight-systems> [Accessed 26 October 2022].

Petikam, L., Anjyo, K. and Rhee, T., 2021. Shading Rig: Dynamic Art-Directable Stylised Shading for 3D Characters. *ACM Trans. Graph.*, [online] 40(5). https://doi.org/10.1145/3461696.

*Spline Tool - Unreal Engine 4 Tutorial - Easy Way to Create Ropes and Chains*. 2020. Available at: <https://www.youtube.com/watch?v=CLPTs2vGb08> [Accessed 26 October 2022].

*Splines are Awesome!!!*. 2020. Available at: <https://www.youtube.com/watch?v=7j\_BNf9s0jM> [Accessed 26 October 2022].

SVG Group, 2022. *Paths — SVG 2*. [online] Available at: <https://www.w3.org/TR/SVG2/paths.html#PathDataCubicBezierCommands> [Accessed 27 October 2022].

Unity Technologies, 2022a. *Unity - Manual: GPU Usage Profiler module*. [online] Available at: <https://docs.unity3d.com/Manual/ProfilerGPU.html> [Accessed 26 October 2022].

Unity Technologies, 2022b. *Unity - Manual: Writing shaders overview*. [online] Available at: <https://docs.unity3d.com/Manual/SL-ShadingLanguage.html> [Accessed 1 November 2022].

YivanLee, 2022. *虚幻4渲染编程(Shader篇)【第十七卷：Geometry shader in UE4】*. [online] 知乎专栏. Available at: <https://zhuanlan.zhihu.com/p/100834351> [Accessed 27 October 2022].